1 TODO

- ☐ CHeck function and procedure signatures when calling them
- ☐ Confirmed A procedure can be called withouth parameters. Please handle that appropriately
- ☐ Register the parameters for program in the global symbol ssope
- ☐ Handle NOT factor
- ☐ What about negative variables
- ☐ Handle if statements
- ☐ Add line num to every node
- ☐ Manage funcction and procedure calls. Remember that if they dont have parenthesis it means they are not passing argumetns
- ☐ SUEPR TODO handle array parameters. And for any array handle multiple dimensions.
- ☐ Not absolutely necessary but the ast should replace their symbols names with reference to the symbol table that contains the info of the respective symbol. Actually I think this wont be necessary in this project as long as I make sure to free the tree at the very end. If We create the table between the creation of the AST and its termination then we can just point to the strings inside the tree;
- ☐ I am using an unordered list as the symbol table. Maybe change it
  - ☐ Symbol table has a limited amount of entries maybe change that and allocate more dynamically
- ☐ Implicitly declared loop variable???
- ☐ Check the all encompassing pdf thingy
- ☐ Just make sure that stuff that you took reference on in semantics is checked to see if we are missing anything
  - ☐ Such as declaredIdList
- ☐ Check if calling function is passing the right type of parameters
- ☐ Nested Methods with same names. I dont know if they are asked for but they talk about it in page 322 of the book
- ☐ Check if num should either be split into real CONst and int const or add a child node to the NUM constant that specifies taht. Because at the type of doing type checking we need to see if there is incompatibility. But then again we can perfectly have a n integer onstant be a real constant and we could round an integer constant to be a real constant.
- ☐ Check that what we are doing of simply passing arguments to the hande-VarDeclarations thingy is okay and wont cause problems later.

2 Errors

- ☒ For some reason it says that I'm redeclaring function 1
  - There is an error with memory where after redifining a variable in a deepr scope once and later closing the scope and later opening another scope with the same name variabel. THen in theat case

The Outermost name will now have a prevHash when it didnt before. COrrect behavior is not to have it since the outermost is the one that is active right now
  - Fixed it by copmletely changint the function that deletes stuff and using double pointer as I shouldve from teh beggining
- ☒ Params is not storing the itneger pointer. I was just passign th epointer for idlist instead of for the actual ids. Nice
- ☒ Redeclarations stop the program
  - Im just adding them to the table but i dont know 100% if its correct.

3 Notes

- A symbol table is responsible for tracing which declaration is in effect when a reference to the symbol is encoutnered.
- Cuando llamas una funcion en lambda "function max(num1, num2: integer): integer;" lo haces de esta manera " max(a, b);". Osea no definis un alista en los parametros
- I dont think we have to implement funciton signatures but i found their definition : TYpe signature of a method includes thenumber and types of its parameters and its return type
- Creo que el primer hash found in the hash chain va a ser el que esta en el scope immediatamente de arriba
- Expression are done by using binary expressions and unary expresions. Each Willspecify their:
  - type
  - operator
  - subexpression
- TYpe declarations are talked about in section 8.6 and 8.8

3.1 Verty Important Notes From the txt File00

1. No new type definitions.
2. They stick with the fact that numb can be both an integer and a string like 2.53E+24
3. WE CANNOT:
   1. ADD
   2. SUBTRACT
   3. DIVIVDE
   4. MULTIPLY Reals and ints together.
4. I dont knwo if this applies to this project but it says here that afunction's value is the value of the variable whose name is the same as the function. `function addition(a, b: integer) : integer; begin addition := a + b; // this is the return value end;`
5. We can assign an array to another `var a, b: array [ 23 .. 57 ] of integer; a := b;` 1.(I do not know if this aplpies): All parameters

are passed by value

6. We may assign an array to another `VAR a, b : array [ 1 .. 10 ] of array [ 1 .. 10 ] of Integer;  a[5] := b[3];`
7. Array indices *CAN* be negative. (Yeah *CAN* not *CAN'T*)
8. We may call a function with or without parameters. (opf course according to their definition)
9. SUPER TODO sFunction and Variable may have same names aslong as return types are differeng
10. Mini pascal is *NOT* case sensitive. Creo que esto ademas de keywords tambien significa Var abc = Var AbC?
11. "You may add overload resolution to the compiler"???

4 References

1. yacc - How Compiler distinguishes minus and negative number during parser process - Stack Overflow