Design Document

Carlone & Nordby

The grammar for this document and project can be found here.

list.h & list.c

The file to create a list structure. The use of this will primarily be in the symbol table generated by the compiler. Functions for this structure include:

1) makelist - allocates memory space for a new list node and assigns the name and sets the pointer to the next node to null.
2) free_list - will free up the memory allocated to a list when the list should be destroyed.
3) list_insert - insert a list node to the top of a list
4) list_search - searches through the list to find a node
5) list_length - returns the amount of nodes in a list

semantic.h & semantic.c

This file is primarily used to store functions that will do the semantic checking for a program to make sure that it fits with the language. This will be used by the parser to make sure there is correctness in the program trying to be compiled. Functions in this file include:

1) semantic_lookup - checks to see if an identifier exists in the scope.
2) semantic_set_type - sets all types in tree to specified type value in a list of identifiers
3) type_of - returns the type of a tree

4) semantic_set - will return if an identifier has already been declared, and will insert the identifier into the scope if it has not been declared.
5) int_to_type - will consider the type of an integer and convert that into the meaning of the integer. This could be an integral, rational, type error, add operation, and so on.
6) type_check - Will kill the compiler if there is a type mismatch.
7) double_check - Will kill the compiler if an identifier is trying to be inserted into a scope in which it has already been declared
8) declare_check - Will kill the compiler if the identifier being declared is being placed in a null list. JONATHAN WHAT IS GOING ON WITH THIS ONE?

symtab.h & symtab.c

This file contains files necessary to interact with the symbol table of the program. The functions for interactions include:

1) make_scope - allocates memory for a new scope as well as initially sets values to null.
2) free_scope - deallocates memory for a scope
3) scope_push - pushes a new scope on top of previous scopes
4) scope_pop - removes the top scope of the program
5) scope_insert - this function will hash the name into the index of the scope and will insert the hashed name into the table
6) scope_search - searches for an identifier in one scope with the help of the hash function.
7) global_scope_search - searches for an identifier in all scopes.
8) hashpjw - function to hash identifier names into the index used for the scopes
9) scope_print - print the index and information in a table within a scope.

tree.h & tree.c

This file is used to create the structure of a tree. This is important in the project for constructing the syntax tree of a program. It is also the host of the assembly code generation. The functions in this file includes:

1) make_tree - allocates memory for a tree and initializes the type, left subtree, and right subtree.
2) make_id - this is a function used for specifically creating a tree for an identifier.
3) make_inum - this is a function used for specifically creating a tree for an integer value.
4) make_rnum - this is a function used for specifically creating a tree for a rational value.
5) return_scan - will scan a tree for an identifier and return a 1 or 0 based on if the identifier was found.
6) print_tree - will print the tree using help from the aux_print_tree function
7) aux_print_tree - This function will print out the tree in a fashion that will space out parents and children as if they were drawn on paper.
8) eval_tree - this will evaluate a tree with simple arithmetic.
9) rank - gives a ranking to leaves on the tree.
10) gencode_start - starts the code generation by opening the target file and creating the data needed for the code generation to continue.
11) gencode - generates assembly code by calling the code generation for statements.
12) gencode_statements - creates the assembly code for statements of the program.
13) gencode_expression - code generation for the expressions of the program.

qc.y

The yacc file that will act as the parser for the compiler.

qc.l

The lex file that will act as the scanner for the compiler.