CS2002 W09 Practical 160015074 Tutor: Tom Kelsey Date: 10/3/2017

Overview:

The objective of this practical was to create a program capable of parsing a specification for a circuit of gates and simulating its outputs when given certain inputs.

Design:

The main tasks of the practical are creating a parser capable of reading a file's worth of Gate definitions and creating their corresponding gates, as well as functions capable of simulating the behaviour of the circuit based on changes to input values.

The parseStream function in streamParser.c is responsible for reading from the standard input stream. It tokenises each line and passes on the full set of tokens for a line to the makeGate function to create a Gate structure representing the gate defined on a line. It repeats this process until there are no lines of input left to read. Each Gate is added to a Circuit structure created earlier, which is returned to whichever function called parseStream in the first place.

The makeGate function in the Gate.c file is responsible for the creation of Gate structures. It takes four parameters, representing the new Gate's name, operation type, the name of the gate providing its A input and the name of the gate providing its B input. The function then allocates memory for a Gate and begins copying each of the given strings into the Gate's fields, before returning the new Gate structure. Later in Gate.c are functions that hold the logic of each Gate's operation – pointers to these functions are also inserted into each new Gate once its type is determined, so that a Gate can be evaluated by passing variables into a function retrievable from inside the very Gate being evaluated, instead of performing another costly type determination.

The List.c file contains function related to the management of a headered linked list. The most relevant function is the addNode function, which determines if a list has been unused and so initializes the head and tail pointers in the header with the new node or appends the new node to the tail of the list.

Circuit.c contains functions related to the creation, management and evaluation of a circuit of gates. The makeCircuit function initialises two lists, one for tracking all created gates including input gates, and another for only tracking input gates. This is intended to make gate linking operations simpler, by only needing to operate on a single list without need for list concatenation or searching multiple lists.

The linkGates function iterates through the complete list of all gates and analyses each stated input for a gate. This is accomplished thanks to the name of a gate's input gates being copied into the gate when it was first parsed. The copied names are then used to search the list of gates, and, when the relevant gate is found, the pointer that held the name of the input gate is reused and reassigned to the input gate structure itself. This operation must be performed after input of all gate specifications is complete to ensure that any inputs that are referred to have been created.

The remaining important function in Circuit.c is evaluateCircuit, which checks whether a given Circuit ever stabilises for the inputs it has. This function assumes that all gates are evaluated simultaneously and evaluates the output of each gate. The inputs to each gate

are only updated once each iteration of evaluation is complete. A circuit is deemed to stabilise once outputs of each gate are the same for two evaluations in a row. A circuit is deemed as impossible to stabilise when the upper bound of 2 to the power of gates is reached, which is the maximum number of unique output combinations a circuit can have.

The main function leverages each of the functions described above to evaluate a circuit that the program is given. In order to generate each possible combination of input values, each input value is equated to a bit in an integer with total bits equal to the number of inputs, with the integer used ranging from 0 to 2^total input gates – 1, and each whole number increase in the integer representing a new set of inputs. This ultimately results in a set of for loops manipulating an unsigned long long through bit shifting. The main function also prints each input value and output value for user viewing.

Compilation:

The program can be compiled by providing all evident .c files to gcc and running it with the -lm flag to allow linking in mathematics libraries. A makefile is also provided, with default, verbose and debug options. By default, running “make” or “make circuits” will not print any warnings.

Testing:

The program was tested throughout using print statements inserted in strategic locations to determine the program’s state at relevant points in time, as well as by checking the program state by hand with GDB. Once the program was near to totally complete, valgrind was used to determine memory usage and diagnose leakages. Once the program was assumed feature complete, testcases were created and the program run against them. An example testing script named runTest.sh is provided with this submission, however it will be necessary to modify the permissions of the script to run it. The testing files used by runTest.sh can be found in the directory “./testfiles/”.

Evaluation:

The program manages to interpret and generate a circuit of gates successfully and correctly. An effort was made to separate and group functions into different files to make updating and bug fixing easier, at the expense of several source files. The location of some structure mallocs is questionable (see ListHeader mallocs in makeCircuit, instead of a dedicated function). Further, the inputs list in a Circuit is never anything other than a subset of the gates list, which is a somewhat needless duplication, meaning a single list or two but-entirely-unique lists solution would likely be a cleaner implementation.

Conclusion:

The program completes the basic specification, performs acceptably, and passes both stacscheck and custom tests.

Extension:

As an extension I have modified the loop in the main method to preserve wire state across different input sets. This has the effect of allowing circuits that would not otherwise stabilise to stabilise, as well as showing the effects of gradual modification of a circuit on its state.

CS2002 W09 Practical 160015074 Tutor: Tom Kelsey Date: 10/3/2017

The program has also been extended so that it handles erroneous Gate specifications better, skipping over Gates with too few inputs, and stops completely if it cannot link the given gates together, perhaps due to a non-existent input being specified.

An extension executable can be created with “make extension” in the root submission directory, or “make” in the “extension” folder.