Naive Implementation of tob

Code Documentation

Faithful Pseudo Code

We only commit two global variables **string error** and **unordered map prio : STRING → INT × CHAR**, both initialized to null.

error is used to capture errors, globally. If it's nonempty and we pass into the main loop, the program will stop and print **error**.

prio[x].first is the priority number $\boldsymbol{x}$ is resolved; **prio[x].second** represents which algorithm to use when restoring/removing parentheses. The lower priority numbers are serviced first, following the shape specified by the character, and with maximum reach.

We define **struct node {char val, node* next, node* left, node* right}**. We want to express strings of *mathematical alphabets*. A mathematical alphabet itself is captured as a string of **char** types. For example $\boldsymbol{x_7}$ is regarded as an individual mathematical alphabet (distinct from $\boldsymbol{x}$ or $\boldsymbol{x_{6+1}}$); we represent it by **x_7**.

MAIN()

GOAL. Runs starting instructions and begins main loop.

main():

1. **Startup()**
2. **main loop()**

STARTUP()

GOAL. Checks for files and reacts accordingly. Initializes **prio**.

Startup():

1. Re-creates "./root/buffer.txt"

2. Verifies files in "./root/", creating them if necessary. "./root/main.txt" is an exceptional case; it's absence is judged as a critical error.

3. Verifies "./Precedence/main.txt" and calls **make prio()**.

4. Prints the first line of "./root/main.txt" – the theory's name.

MAKE PRIO()

GOAL. Initializes map **prio** from "./Precedence.main.txt". Each row of the text file is a concatenation of **"A B"** where **A** is a string, **B** a char, and **A** and **B** are separated by white-space. **prio** assigns $\boldsymbol{A} \rightarrow$ (row number$\boldsymbol{,B}$).
Exceptions:
1. These strings can't include **_**, white-space, or parentheses.
2. A string can appear, at most, once in the file.
3. The pairing char must be found in **QBbUuNn**. **Q** quanified, **B** second-order binary, **b** first-order binary, **U** second-order unary, **u** first-order unary, **N** second-order nullary, and **n** first-order nullary; characters outside the precedence and outside the logical apparatus are treated as **n**.

make prio():

1. Empties **prio** and opens "./Precedence.main.txt".

2. In reading each row of the file, we first scan for the first whitespace; if the character to follow the whitespace is one of the special characters, and if the string is unmapped, we populate **prio** with the corresponding assignment. Otherwise we assign **error** appropriately and return.

MAIN LOOP()

GOAL. Continually reads user input and executes the corresponding function.

make loop():

1. while **error** is empty and **getline(cin, input)**:
   (a) if input is of form **"help "** + ***X***, do **help(X)**.
   (b) elif input is **_quit**, set **error** to something non-empty.
   (c) elif input is **show prec**, print **prio**.
   (d) elif input is **show sig**, print "./root/sig.txt".
   (e) elif input is **add prec**, **add prec()**.
   (f) elif input is **del prec**, **del prec()**.
   (g) elif input is of the form **"add axm "** + ***X***, **add axm name(X)**.
   (h) elif input is **show logs**, **show logs()**.
   (i) otherwise print invalidity.
2. Print **error** and return.

HELP(CONST STRING& TAG)

GOAL. If **tag** is empty, this function prints a list of commands, otherwise if **tag** is a command, it prints more pertaining information. (Note this prevents impredicative arguments, ie. **tag** cannot self-reference **help**.)

help(const string& tag):

1. If **tag** is empty, display the list of commands: **_quit**, **show logs**, **show prec**, **show sig**, **add prec**, **del prec**, **add axm <file>**, **help <command>**.
2. If **nonempty** is one of the commands, print additional information. Otherwise print invalidity.

ADD PREC()

GOAL. Adds an element to map **prio**. Accepts only inputs of the form **A_B_C** where **A** is the string, **B** the int, and **C** the char. If **A** is legitmate (eg. unmapped), **B** within range, and **C** among the accepted chars, we add **A** → **(B,C)**. If **B** were to have trailing **0**s, we insert the element after shifting all rows greater than or equal to **B** by one. These changes take place in "./Precedence/main.txt".

add prec():

1. While **getline(cin,** input**)** and input does not equal **_quit**:
   (a) Scans input for the first occurrence of _ and saves the resulting substring. If any errors come up (eg. if **strings disjoint("(_)",** string**)** is false), then it prints the problem and **continues**.
   (b) Scans input for the next occurrence of _, **stoi()**s the string inbetween, and checks consistency with the current precedence. On leading **0**s, **bool insert** is made true. Prints errors (eg. **strings joint("0123456789",** string) is false) and **continues**, if any.
   (c) Scans the final portion, confirming its an accepted char. On errors, we print and **continue**, otherwise we **break**.
2. If the input were **_quit**, return. Otherwise we add the inputted element to **prio** and store the lines of "./Precedence/main.txt".
3. If **insert** is false we simply increment the line by the new element, at the corresponding row, and re-write the file.
4. Otherwise we place the new element in its row, standalone, and then write the remaining lines afterwords. We call **make prio()** to update.

BOOL STRINGS DISJOINT(CONST STRING& A,B)

GOAL. Returns true iff. the intersection between the set of characters involved in **a** and the the set of characters involved **b** is empty

bool strings disjoint(const string& a, const string& b):

1. Populates, via hash-functions, an initially empty map, **map : CHAR → BOOL**, with elements **(i, true)** for each char ***i*** in ***a***.

2. If any **i** in **b** is in the domain of **map**, then return false. Otherwise, after the loop, return true.

BOOL STRINGS JOINT(CONST STRING& A,B)

GOAL. Returns true iff. the set of characters involved in **a** is a subset of the set of characters involved in **b**.

bool strings joint(const string& a, const string& b):

1. Populates, via hash-functions, an initially empty map, **map : CHAR → BOOL**, with elements **(i, true)** for each char ***i*** in ***a***.

2. If any **i** in **b** is not in the domain of **map**, then return false. Otherwise, after the loop, return true.

DEL PREC()

GOAL. Deletes an element from map **prio**. Accepts only mapped strings. If we were to delete the last item in a row then we lower all higher rows by one. These changes take place also in "./Precedence/main.txt".

del prec():

1. Loops user input until either it is **_quit** or a mapped string.

2. If the input were **_quit**, return. Otherwise we delete the inputted element from **prio** and store the lines of "./Precedence/main.txt".

3. If we didn't delete the last item in the row, we simply scan that line for the element, remove it, and re-write the file.

4. Otherwise we remove the line entirely, and re-write the remaining lines on top. We call **make prio()** to update.

ADD AXM NAME(STRING TAG)

GOAL. Adds an axiom to the theory by name of **tag**, if appropriate. Should the attempted added axiom be syntactically correct (ie. in the underlying formalism the string added must be an *object*), a corresponding file is created in"./Axioms/". The two files "./root/sig.txt" and "./root/logs.txt" are updated.

add axm name(string tag):

1. Returns if **tag** is invalid (ie. when **tag** is empty, when **strings disjoint("_", tag)** is false, or when **dir hasfile**("./Axioms/**tag**.txt") is true).
2.