P0

Due Feb 23 by 11:59pm **Points** 100

Programming refreshment, practice with standards and header files, practice with trees, traversals, command line arguments, and file IO.

Note: command line arguments and keyboard input can directly be reused in the later compiler project so make sure to do it well and modular.

Submission:

/accounts/classes/janikowc/submitProject/submit_cs4280_P0 *SubmitFileOrDirectory*

Write a program to build a tree and print it using different traversals. The program will be invoked as

P0 [*file*]
where *file* is an optional argument.

If the *file* argument is not given, the program reads data from the keyboard as a device (15%).

If the argument is given, the program reads data file ***file*.sp2020**. (note that *file* is any name and the extension is implicit).
Programs improperly implementing file name or executable will not be graded. **15% is for style**. **15% interactive input**. **70% performance**.

Example invocations:

P0 // read from the keyboard until simulated EOF
P0 < *somefile* // same as above except redirecting from *somefile* instead of keyboard, this tests keyboard input
P0 *somefile* // read from *somefile*.sp2020

- Assume you do not know the size of the input file
- Assume the input data is all strings separated with standard WS separators (space, tab, new line)
- If the input file is not readable for whatever reason, or command line arguments are not correct, the program must abort with an appropriate message
- The program will read the data left to right and put them into a tree, which is a binary search tree (BST) with respect to the string length , that is all strings of the same

length are falling into the same node in the BST.
- Tree is never balanced nor restructured other than growing new nodes.
- A node should contain all data that falls into the node except that literally the same strings will show up only once (the node contains the set of data falling into the node).
- The program will subsequently output 3 files corresponding to 3 traversals, named ***file*.preorder**, ***f*ile.inorder** and ***file*.postorder**. Note that ***file*** is the base name of the input file if given, and it is **output** if not given.
  - Treversals
    - preorder
    - inorder
    - postoder
  - Printing in traversal
    - a node will print the node's size (the length of the strings in the node) intended by 2 x depth of the node, followed by the list of strings from the node set (the root is considered level 0)
- Example will be elaborated in class

  File **xxx.sp2020** contains

  adam ala jim ada john ala susana george gorge duck george asa james 123 12
  - invocation: > **P0 xxx**
    - Output files: **xxx.inorder xxx.preorder xxx.postorder**
  - Invocation: > **P0 < xxx.sp2020** // or > **P0**, followed by typing the data followed by EOF
    - Output files: **ouput.inorder output.preorder output.postorder**
- Standards related requirements:
  - Have the following functions/methods minimum (the types and arguments are up to you, the names are required as given)
    buildTree()
    printInorder()
    printPreorder()
    printPostorder()
  - These must be in one file called **tree.c** or appropriately based on your language, with tree.h or as appropriate for your language
  - Define the node type in **node.h**
  - Keep the main function in a separate file (regardless of language)

Traversals taken from the 3130 textbook:

Preorder:

- process root
- process children left to right

Inorder:

- process left child
- process root
- proccess right child

Postorder:

- process children left to right
- process root

More suggestions

- Using top-down decomposition you have 3 tasks in main:
  - *Process command arguments*, set up file to work regardless of source, check if file readable, set the basename for the output file, etc.
  - *Build the tree*
  - *Traverse the tree* three different ways generating outputs

The main function should handle the 3 functionalities. #1 should be handled inside of main, functions for #2 and #3 should be in another separate source as indicated. Any node types should be defined in a separate header file.

For development purposes, do either 1 or 2 first. 3 should follow 2, first with one traversal only.

Processing either keyboard or file input can be done one of these two ways:

1. If keyboard input, read the input into temporary file, after which the rest of the program always processes file input
2. If keyboard input, set file pointer to stdin otherwise set file pointer to the actual file, then process always from the file pointer

Files:

- **node.h**, **tree.c+tree.h** (or as appropriate for your language)
- **main.c**,
- **makefile**

Sample structure for main.c

```
#include "node.h"
#include "tree.h"
int main(int argc, char* argv[]) {
  // process command line arguments first and error if improper
  // if filename given, make sure file is readable, error otherwise
  // set up keyboard processing so that below this point there is only one version of the code
```

```
node_t *root = buildTree(file);
printPreorder(root);
printInorder(root);
printPostorder(root);
return 0;
}
```

Ideas for printing tree with indentations

```
static void printPreorder(nodeType *rootP,int level) {
  if (rootP==NULL) return;
  printf("%*c%d:%-9s ",level*2,' ',level,NodeId.info); // assume some info printed as string
  printf("\n");
  printPreorder(rootP->left,level+1);
  printPreorder(rootP->right,level+1);
}
```

Testing

This section is non-exhaustive testing of P0

1. Create test files:
    1. P0_test1.sp2020 containing empty file
    2. P0_test2.2020 containing one string: adam
    3. P0_test3.sp2020 containing some strings with same length (some repeats) and different length over multiple lines, e.g., use the example from above
2. For each test file, draw by hand the tree that should be generated. For example, using P0_test2.sp2020 should create just one node printing: 4 adam
3. Decide on invocations and what should happen, what should be output filenames if no error, and what the output files should look like - using the hand drawn trees for each file
4. Run the invocations and check against predictions
    1. **P0 < P0_test1**
       Error
    2. **P0 < P0_test3.sp2020**
       Outputs output.inorder output.preorder output.postorder, each containing the stings (no repetitions)
    3. **P0 P0_test3**
       Outputs P0_test3.inorder P0_test3.preorder P0_test3.postorder containing as above

Grading

- 15% standards, coding and architectural
- 15% keyboard input

- 10% proper errors
- 60% proper outputs