C 課題 PJT (Perfect Jouraku Tree)

次の仕様 (1~3) を満たすコマンド PJT を作成する。

1　ファイルまたは標準入力から入力を読み込む

- コマンドラインに 1 つ以上の引数 (後述するオプションを除く) が与えられた場合、それらを入力ファイルとして順に読み込む。空ファイルを読み込んだ場合は読み込み ERROR(NULL 判定) が出ない限り標準入力からの読み込みはしない。読み込み ERROR の場合はエラーを出力するようにする。
- 引数 (後述するオプションを除く) が 1 つも存在しない場合、標準入力からの入力を読み込む。

∗ scanf は使用不可。fgets を使用する。
∗ 文字読み込みに関して、space 区切りにも対応させる。改行文字は 1 単語として扱わない。

入力例: 3 行目は 3 単語分、6 行目は単語なし、7 行目は先頭の space を除く

```
%cat hoge.txt
aaa
ccc
bbb dddd ss
as
asfas

  asds
bbb
```

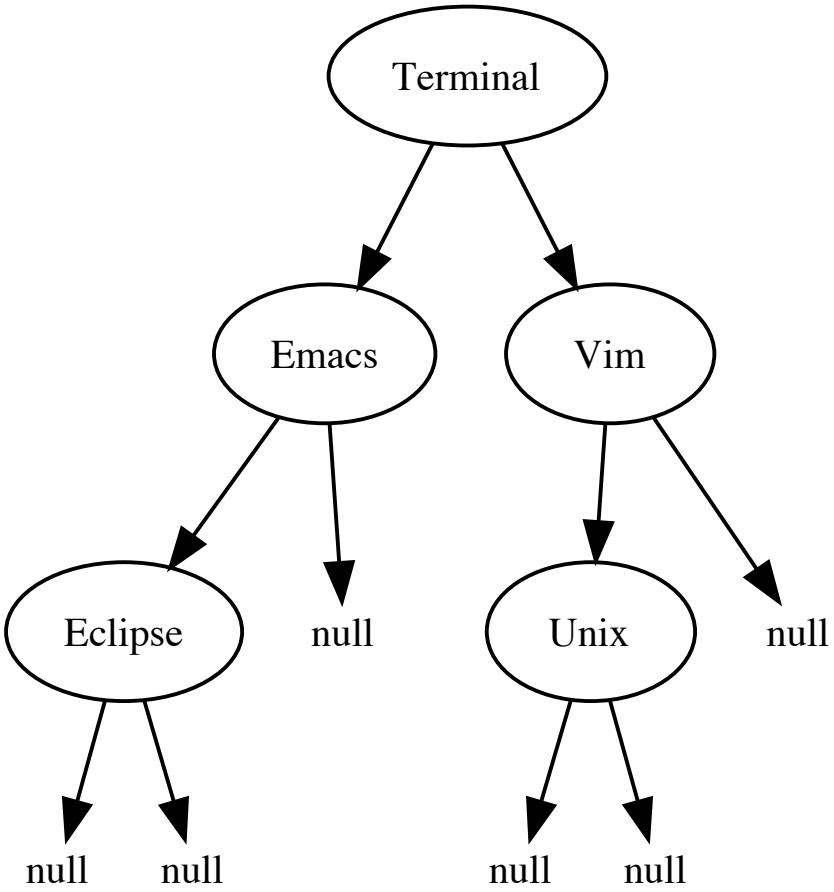
2　入力から単語を順に切り出し、2 分探索木に格納する

2 分探索木について

各ノードの子の数が高々 2 となるような順序木を 2 分木と呼ぶ。各ノードのデータが次の条件を満たす 2 分木を特に 2 分探索木と呼ぶ。

(条件)

- あるノードから見て、左側の子ノードおよびその子孫ノードのデータ全てが自身が持つデータ (文字列) より小さい
- あるノードから見て、右側の子ノードおよびその子孫ノードのデータ全てが自身が持つデータ以上の大きさ



2 分探索木の各ノードは次のデータを持つものとする。

- 文字列 (入力内容の単語, 英数字)
- 左側の子ノードへのリンク
- 右側の子ノードへのリンク
- 親ノードへのリンク

* リンク先のノードが存在しない場合は、それぞれ NULL を持つ
* 2 分探索木の各ノードを表す BinSTreeNode 型について

```
typedef struct _BinSTreeNode{
    char *word;
    struct _BinSTreeNode *left;
    struct _BinSTreeNode *right;
    struct _BinSTreeNode *pare;
}BinSTreeNode;
```

で統一する。また、root node の名前は

```
BinSTreeNode *rootNode
```

で統一する。

3 作成した 2 分探索木に対して、コマンドラインで指定されるオプションに応じた処理を実行する

1) **オプションなし (デフォルトの動作)**
生成した 2 分探索木の各文字列を間順走査で表示する (つまり次の-p 1 と同義)。

2) -p [number] (0:前順走査 1:間順走査 2:後順走査)
生成した 2 分探索木の各文字列を指定された走査順で表示する。引数が存在しない場合は number=1 として間順走査をさせる。

-p オプションを指定した場合の例

```
%cat testcase/1abc.txt
defg
abc
zzzz
whasiuf
y
goo
pin
hanpen
%./pjt -p 1 testcase/1abc.txt
---BinSTree---
abc
defg
goo
hanpen
pin
whasiuf
y
zzzz
```

3) -s /match/replace/

生成した 2 分探索木の各ノードが持つ文字列について、部分文字列 match を含む場合、文字列 replace に置換する。replace に space が 1 つでも含まれる場合はオプションエラーを返す。また、replace が空文字の場合は、部分文字列 match を削除する。

-s オプションを指定した場合の例

```
%cat testcase/subst_sample.txt
defg
hoge
zzzz
defanddef
%./pjt -s /def/apple/ testcase/subst_sample.txt
---BinSTree---
appleandapple
appleg
hoge
zzzz
```

-s /match//とした場合の例

```
%cat testcase/u_sample.txt
hoge
hage
hoge
hage
onemu
%./pjt -s /ge// testcase/u_sample.txt
---BinSTree---
ha
ha
ho
ho
onemu
```

4) -u

2 分探索木に同じ文字列が複数存在する場合 1 つのみを出力する。引数は取らず、走査順は-p オプションに従う。

-u オプションを指定した場合の例

```
%./pjt -u testcase/u_sample.txt
---BinSTree---
hage
hoge
onemu
```

5) -r /removematch/delall/

2 分探索木から与えられた文字列 removematch を持つノードを削除する。delall が真 (0) であれば、全て削除し、偽 (0 以外) であれば root ノードから一番近いノードのみを削除する。

-r オプションを指定した場合の例

```
%./pjt -r /hoge/0/ testcase/u_sample.txt
---BinSTree---
hage
hage
onemu
%./pjt -r /hoge/1/ testcase/u_sample.txt
---BinSTree---
hage
hage
hoge
onemu
```

オプション補足

- 文字列が空になったノードは削除する。
- option が 1 つでも間違っていた時点で program は動かさず、正しい使い方のヘルプを表示させる。
- '-' から始まる引数は全てオプションとして扱う。

- 入力ファイルは option の後にしか存在しない設定とする。

コマンド実行時の例

```
（正）%./pjt -r /match/delall/ -u -p 0 -s /match/replace/ file1 file2 file3
     %./pjt -p 0 -u file1
（誤）%./pjt -r /match/delall/ -u file1 file2 file3 -p 0 -s /match/replace/
     %./pjt -p file1 -u file2
```

- -s,-r オプションに関しては複数回実行できるようにする。また、その際与えられた option の順番に応じて実行するようにする。queue の実装推奨。

-s, -r オプションを複数回指定した場合の実行例

```
%./pjt -s /ge/go/ -r /hogo/1/ -s /hogo/banana/ testcase/u_sample.txt
---BinSTree---
banana
hago
hago
onemu
```

- -s,-r オプションによって、ノードが全てなくなった場合はなくなったことを表示するようにする。

ノードがなくなった場合と、標準入力を行った場合の実行例

```
%./pjt -r /hoge/0/
---standard input---
hoge
hoge
hoge
---BinSTree---
There is no node
```

- オプションやファイル名が不適切な場合はエラーを出力する。また、何が不適切だったのかをエラーメッセージとして出力する。

指定したオプションが不適切な場合の実行例

```
%./pjt -r /a/
usage:ERROR (incorrect -r format follow -r /rmvmatch/delall/)
%./pjt -d
usage:ERROR (options are invalid)
%./pjt non_existing_file.txt
usage:ERROR (File is NULL(read file))
```

上記の仕様を実現するために 2 分探索木に対して操作を行う関数を作成する

以下に記す関数は必ず作成する。また、各関数は最低限、次の機能を持つ。

1) BinSTreeNode* createNode(const char *word)

引数 word で与えられた単語 (文字列) を持つ (2 分探索木の) ノードを動的に生成し、生成されたノードを返す (失敗した場合には NULL を返す)。与えられた文字列についても、動的にメモリを確保する。

(引数)

`const char *word` : **単語を表す文字列**

(返り値)

`BinSTreeNode*` : **動的に生成されたノード (生成に失敗した場合は** `NULL` **を返す)**

2) void addNode(BinSTreeNode *rootNode, BinSTreeNode *node)

2 分探索木に対してノードを新たに追加する。

(引数)

(1) `BinSTreeNode *rootNode` : **操作対象となる 2 分探索木の** `root` **ノード**

(2) `BinSTreeNode *node` : **追加するノード**

(返り値)

なし

3) void printTree(BinSTreeNode *rootNode, int order)

引数 order で指定された走査順に基づいて、与えられた 2 分探索木の各ノードの文字列を順に表示する。

（引数）

（1）`BinSTreeNode *rootNode` : **操作対象となる 2 分探索木の** `root` **ノード**

（2）`int order` : **2 分探索木の走査順を指定する整数**

（返り値）

なし

4) int substString(BinSTreeNode *rootNode, const char *sstr, const char *rstr)

2 分探索木の各ノードについて、引数 sstr で与えられる文字列が各ノードが持つ文字列の部分文字列となっている場合、その部分文字列を引数 rstr で与えられた文字列で置換する。返り値として、置換を受けたノードの数を返す。なお、部分文字列の置換を行った場合、2 分探索木の条件が崩れる可能性があるため、置換後に後述の sortBinSTree 関数を実行する。

（引数）

（1）`BinSTreeNode *rootNode` : **操作対象となる 2 分探索木の** `root` **ノード**

（2）`const char *sstr` : **置換の対象となる部分文字列**

（3）`const char *rstr` : **置換後の文字列**

（返り値）

`int` : **置換されたノードの数**

5) void sortBinSTree(BinSTreeNode *rootNode)

与えられた木に対して、親ノードと子ノード間のデータの大小関係が 2 分探索木の条件を満たすようにソートする (必要に応じて親ノードと子ノードを入れ替える)。

（引数）

`BinSTreeNode *rootNode` : **操作対象となる木の** `root` **ノード**

（返り値）

なし

6) void removeNode(BinSTreeNode *rootNode, const char *word, int delall)

与えられた 2 分探索木から、引数 word で与えられる文字列を持つノードを削除する。該当文字列を持つノードが複数存在する場合、delall が真 (0) であれば全て削除し、偽 (それ以外) であれば root ノードから最も近いノードのみを削除する。ノードを削除する際には、malloc によって動的に確保されたメモリを解放する。

（引数）

（1）BinSTreeNode *rootNode : 操作対象となる 2 分探索木の root ノード

（2）const char *word : 削除対象となるノードが持つ文字列

（3）int delall : 該当ノードが複数存在する場合

真（0）-> すべて削除

偽（0 以外）-> root ノードに最も近いノードのみ削除

（返り値）

なし

7) void clearBinSTree(BinSTreeNode *rootNode)

与えられた 2 分探索木のすべてのノードを削除する。プログラムにて 2 分探索木の利用が完了した後で呼び出す。

（引数）

BinSTreeNode *rootNode : 操作対象となる 2 分探索木のノード

（返り値）

なし

コマンドを作成する上での注意

- main 関数と 2 分探索木操作用関数はそれぞれ別のソースファイルに記述する。
- コンパイル時には、-Wall オプションを必ず付けて warning をゼロにする。
- プログラムは出来る限り可読性の高いコードを書く (ex. 適宜コメントを付けて動作を把握しやすいようにする, 変数名の決め方 etc...)。
- メモリリークが全く起きないようにする。
- 2 分探索木に対して操作を行った結果、2 分探索木の条件が崩れないようにする。