数据结构课程设计二实验报告

学号：09021227 姓名：金桥 实验类型：必做

2022 年 12 月 6 日

1. 问题描述：描述实验内容和要求以及需要解决的问题。

提示：结合教师课堂讲授内容，仔细分析实验要求，对实验内容进行需求分析。

问题描述：

- 在 ZIP 压缩文档中，保留着所有压缩文件和目录的相对路径和名称。
- 当使用 WinZIP 等软件打开 ZIP 压缩文档时，可以从这些信息中重建目录的树状结构。

实验要求：

- 实现目录的树状结构的重建。
- N: 取值 $1 \sim 100$ ，表示 ZIP 压缩文档中的文件和目录数量。

输入文件格式：

第一行：N : 文件和目录的数量

第二行：文件或目录的相对路径和名称，每行不超过 260 个字符

输出文件格式：（假设: 所有路径都是相对于 root 目录，每级目录比上一级目录多缩进 2 个空格）

从 root 开始输出每个目录名称

按字典序输出所有子目录

按字典序输出所有文件

注意：

- 文件名称与目录名称，可能存在重名，必须进行区分。
- 路径名称与文件名称，仅包括英文字母，且字母区分大小写。
- 符号‘\’仅作为路径分隔符, 且目录以‘\’结束。
- 不存在重复的输入值。
- 每个文件名后一对“()”内的值，为该文件的大小。

2. 算法思想：详细描述解决相应问题所需要的算法设计思想。

采用**树状数据结构**实现，目录作为父节点，目录下的文件夹以及文件作为子节点。

单个节点存放节点名称，结点类型（文件夹/文件），以及子节点指针。

添加节点时，依此访问其父节点，若其父节点不存在则创建其父节点。创建叶子节点时注意判别是否重复。

在打印之前对所有节点的子节点指针序列进行字典序排序。最后进行输出。

树内部的函数大部分采用递归方式实现。

除此以外，还需要自行实现 `std::vector` `std::string` `std::sort` 的功能，前两者为**顺序表**，后者为**冒泡排序**。

3. 功能函数：描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

MyVec 类：

MyVec ：默认构造函数，生成一个空的 MyVec.

```
1    /**
2     *  @brief MyVec object constructor
3     */
4    MyVec();
```

MyVec ：给定另一个数组的开始位置与结束位置生成对应的 MyVec

通过传入指向开始元素的指针 begin 与指向末尾元素的下一个位置的指针 end 构建 MyVec.

```
1    /**
2     *  @brief MyVec object constructor, copy from begin to end
3     *  @param begin - begin position
4     *  @param end   - end position
5     */
6    MyVec(DT* begin, DT* end);
```

MyVec ：拷贝构造函数

```
1    /**
2     *  @brief MyVec object copy constructor
3     *  @param vec - copy vector
4     */
5    MyVec(const MyVec<DT>& vec);
```

~MyVec ：析构函数

```
1    /**
2     *  @brief MyVec object destructor
3     */
4    ~MyVec();
```

operator[] ：获取下标 n 的元素

```
1    DT& operator[](int n);
```

pushBack ：向尾部插入新元素

```
1    /**
2     *  @brief push element to the end of vector
3     *  @param element - data to be pushed
4     */
5    void pushBack(const DT& element);
```

sort ：按给定方式排列所有元素

传入比较函数，按照比较函数进行排序，类似于 std::sort 函数的第三个参数。

```
1    /**
2     *  @brief bubble sort
3     *  @param fun - sort order
4     */
5    void sort(bool(*fun)(DT, DT));
```

begin ：获取指向首个元素的指针

```
1    /**
2     *  @brief  Return iterator to beginning
3     *  @retval  - iterator to beginning
4     */
5    DT* begin() const;
```

end ：获取指向最后一个元素的下一个位置的指针

```
1    /**
2     *  @brief  Return iterator to end
3     *  @retval  - iterator to end
4     */
5    DT* end() const;
```

size ：获取 MyVec 大小

```
1    /**
2     *  @brief  Return size
3     *  @retval  - size of vector
4     */
5    int size() const;
```

MyStr 类：

MyStr ： 默认构造函数，生成一个空的 MyStr.

```
1    /**
2    *  @brief MyStr object constructor
3    */
4   MyStr::MyStr();
```

MyStr ： 通过字符串常量构建 MyStr.

```
1    /**
2    *  @brief MyStr object constructor
3    *  @param c - string constant. eg. "str"
4    */
5   MyStr::MyStr(const char* c);
```

MyStr ： 拷贝构造函数.

```
1    /**
2    *  @brief MyStr copy constructor
3    *  @param s - copy string
4    */
5   MyStr::MyStr(const MyStr& s);
```

~MyStr ： 析构函数.

```
1    /**
2    *  @brief MyStr object destructor
3    */
4   MyStr::~MyStr()
```

operator[] ： 获取字符串第 n 个字符

```
1    char& MyStr::operator[](int n);
```

substr : 获取字符串 [start, start+offset) 位置的字串

类似于 std::string 的 substr 函数。例如"abcde" 的 substr(0, 3) 就是"abc"

```
/**
 *  @brief  get substring from [start, start + offset)
 *  @param  start  - start position
 *  @param  offset - substring length
 *  @retval        - substring
 */
MyStr MyStr::substr(int start, int offset) const;
```

length : 获取字符串长度

```
/**
 *  @brief  get length
 *  @retval int - length
 */
int MyStr::length() const
```

operators : 重载的运算符函数，与 std::string 的运算符行为一致

```
friend std::istream& operator>>(std::istream&, MyStr&);
friend std::ostream& operator<<(std::ostream&, const MyStr&);
friend MyStr operator+(const MyStr&, const MyStr&);
friend bool operator==(const MyStr&, const MyStr&);
friend bool operator<(const MyStr&, const MyStr&);
friend bool operator>(const MyStr&, const MyStr&);
```

FileTreeNode 类:

FileTreeNode : 构造函数，给定节点名称与是否为文件夹.

```
/**
 *  @brief FileTreeNode object constructor
 *  @param name     - node name
 *  @param isFolder - true: node is a folder
 */
FileTreeNode::FileTreeNode(MyStr name, bool isFolder);
```

~FileTreeNode ： 析构函数.

```
1    /**
2     *  @brief FileTreeNode object destructor
3     */
4    FileTreeNode::~FileTreeNode();
```

getName ： 获取节点名称.

```
1    /**
2    *  @brief  get node's name
3    *  @retval  - node's name
4    */
5   MyStr FileTreeNode::getName() const;
```

FileTree 类:

FileTree ： 构造函数，构造一颗空的 FileTree 树.

```
1    /**
2    *  @brief FileTree object constructor
3    */
4   FileTree::FileTree();
```

~FileTree ： 析构函数.

通过调用 destructRecursion 递归析构整棵树。

```
1    /**
2    *  @brief FileTree object destructor
3    */
4   FileTree::~FileTree();
```

destructRecursion ： 递归析构函数.

传入当前要析构的子树根节点 node 进行析构。

```
1    /**
2    *  @brief destruct recursion
3    *  @param node - current destructing node
4    */
5   void FileTree::destructRecursion(FileTreeNode* node);
```

add : 通过 MyVec 添加节点.

将字符串分解为父节点列表 prefix，节点名称 name，是否为文件夹 isFolder，传入 addNode 函数进行添加。

```
1    /**
2    *  @brief add node to the tree
3    *  @param pth - the path of file/folder
4    */
5    void FileTree::add(MyStr pth);
```

addNode : 接收上一个函数传来的参数，并添加节点.

添加节点的入口函数，调用 addNodeRecursion 完成节点的添加。

```
1    /**
2    *  @brief add node to the tree
3    *  @param prefix   - prefix of path
4    *  @param name     - name of file/folder
5    *  @param isFolder - true: node is a folder
6    */
7    void FileTree::addNode(MyVec<MyStr> prefix, MyStr name, bool
        isFolder);
```

addNodeRecursion : 递归添加节点函数.

```
1    /**
2    *  @brief add node recursion
3    *  @param node     - current node
4    *  @param num      - current pos in prefix
5    *  @param prefix   - prefix of the path
6    *  @param name     - node name
7    *  @param isFolder - true: node is a folder
8    */
9    void FileTree::addNodeRecursion(FileTreeNode* node, int num,
        MyVec<MyStr> prefix, MyStr name, bool isFolder);
```

sortNode : 对子节点进行排序的入口函数.

```
1    /**
2    *  @brief sort all node by dict order
3    */
4    void FileTree::sortNode();
```

sortNodeRecursion ： 递归的对每个节点的子节点进行排序.

```
1    /**
2    *  @brief sort all node recursion
3    *  @param node - current node
4    */
5    void FileTree::sortNodeRecursion(FileTreeNode* node);
```

print ： 打印树的入口函数.

```
1    /**
2    *  @brief print the tree
3    */
4    void FileTree::print() const
```

printRecursion ： 递归的对每个节点进行打印.

```
1    /**
2    *  @brief print recursion
3    *  @param prefix - the output prefix
4    *  @param node   - current node
5    *  @param isLast - true: node is the last subnode
6    */
7    void FileTree::printRecursion(MyStr prefix, FileTreeNode* node,
       bool isLast) const
```

getOutput ： 获取输出的字符串.

如果节点是文件夹则输出时加上下划线。**如果出现乱码请注释掉里面的 if 语句**

```
1    /**
2    *  @brief  get output string of name, if is a folder then give
         it an underline
3    *  @param  name     - name
4    *  @param  isFolder - true: node is a folder
5    *  @retval          - output string
6    */
7    MyStr FileTree::getOutput(MyStr name, bool isFolder) const;
```

4. 测试数据：设计测试数据，或具体给出测试数据。

提示：要求测试数据能全面地测试所设计程序的功能。

测试数据如下：

测试输入数据一：

```
7
b(115)
c\
ab\cd
a\bc(798)
ab\d(161)
a\d\a(1338)
a\d\z\
```

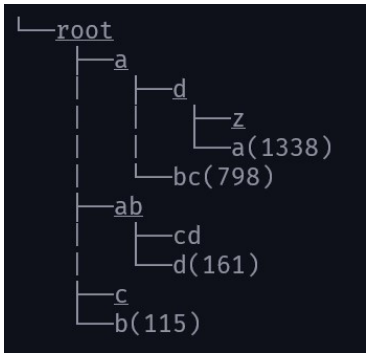
测试输入数据二：

```
22
System\Overview(221)
Abstraction\Notions(183)
Abstraction\Notions\Definition(1248)
Abstraction\Notions\Example(252)
Exercises\
Basice(1956)
OODesign\Evolution\Generations\History\Efficient(14)
OODesign\Evolution\Generations\History\Flexible(11)
OODesign\Evolution\Generations\History\Available(9)
System\Requirements(57)
System\Analysis(233)
OODesign\Decomposition(204)
OODesign\Programming\Object(55)
OODesign\Programming\Programming(48)
OODesign\Programming\Language(52)
System\Design(258)
System\Refinement(137)
System\Verification(451)
OODesign\Evolution\Generations\Language\First(19)
OODesign\Evolution\Generations\Language\Second(20)
OODesign\Evolution\Generations\Language\Third(13)
OODesign\Evolution\Generations\Language\Fourth(24)
```

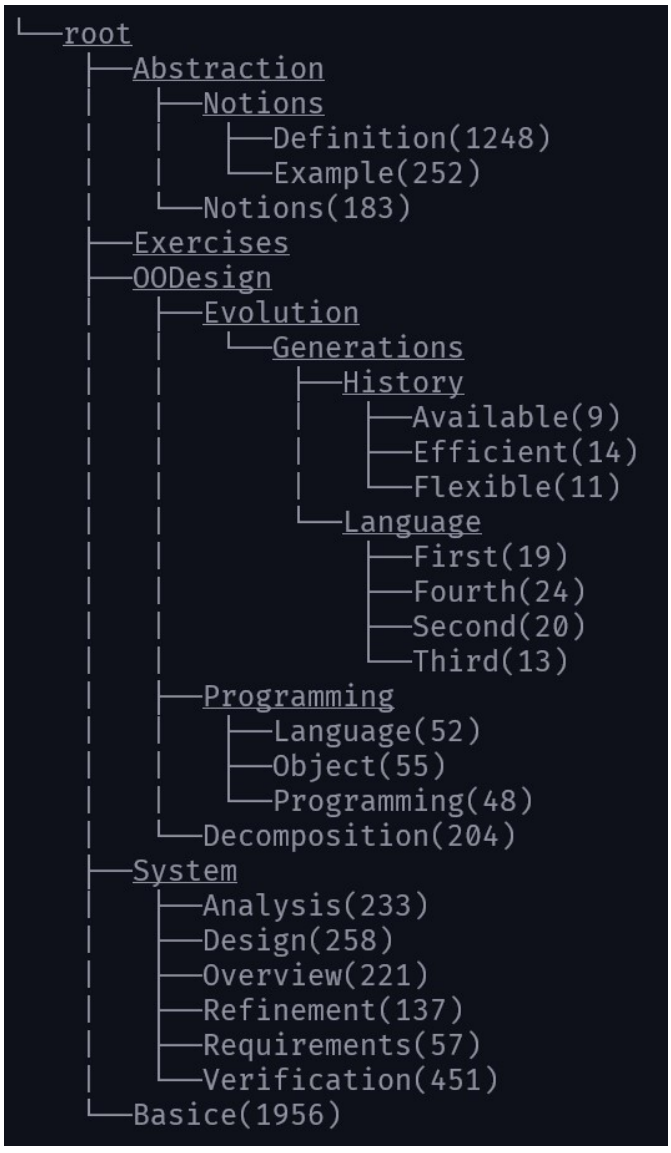
5. 测试情况：给出程序的测试情况，分析运行结果，显示实验结果截图。

实验结果截图：

测试输出数据一：



测试输出数据二：



如图，文件夹带有下划线，文件无下划线。与输入内容预期结果一致。**如果出现乱码请注释掉 `FileTree::getOutput` 函数里面的 `if` 语句块！**

6. 实验总结：写出实验过程中遇到的问题，以及问题的解决过程。分析算法的时间复杂度和空间复杂度，总结实验心得体会。

问题及解决过程：

实验过程无问题。

算法的时间复杂度分析：

`MyVec` 设元素个数为 n，则单次添加为 $O(n)$，访问为 $O(1)$.

`MyStr` 设元素个数为 n，则 `substr` 函数复杂度为 $O(n)$.

`FileTree` 添加单个文件/文件夹：设树的大小为 N，则添加时最坏为 $O(N)$，此时树退化为一条链。

`FileTree` 按字典序排序所有文件/文件夹：设树的大小为 N，使用冒泡排序进行排序，对于一个有 m 子节点的排序时间复杂度为 $O(m^2)$。最坏时树退化为一条链，复杂度为 $O(N^3)$.

`FileTree` 设树的大小为 N，打印树为 $O(N)$.

算法的空间复杂度分析：

`MyVec` 设元素个数为 n 则为 $O(n)$.

`MyStr` 设字符个数为 n 则为 $O(n)$.

`FileTree` 设树的大小为 N 则为 $O(N)$.

心得体会：

感觉 `MyVec` 单次添加的性能还可以优化，类似于 `std::vector` 一样倍增容量而不是每次都重新构建。除了 `iostream` 库以外没有使用任何 C++ 自带内容。

7. 源代码：给出项目所有源程序清单。

建议：源程序中应有充分的注释，例如注释每个函数参数的含义、函数返回值的含义、函数的功能、主要语句段的功能，等等。

实验源程序清单：

1. `main.cpp`
2. `FileTree.h`
3. `FileTree.cpp`
4. `MyVec.hpp`
5. `MyStr.h`
6. `MyStr.cpp`
7. `input.txt`