Michael Schem
Data Structures

Change Log

CategoryMap

The function contains in CategoryMap had a potential memory leak when getting the category for a word. If the code below is used instead there is no dynamic memory to need to delete after the function is done. This is also more efficient because it uses less steps.

```
char* CategoryMap::contains(char word[])
        {
            for (int i = 0; i < size; i++)
            {
                if(strcmp(subCategories[i], word) == 0)
                {
                    return categories[i];
                    break;
                }
            }


    }
```

There is no deconstructor for this class, so the dynamic memory will not be cleared and this will cause a memory leak. To fix this I added a deconsructor as shown below.

```
CategoryMap::~CategoryMap(){
            if(size > 0){
                for(int i=0;i<size;i++){
                     delete subCategories[i];
                     delete categories[i];
                }
                delete [] subCategories;
                delete [] categories;
            }
        }
```

BookParser

The file name is not deleted at the end of the program, this will be a memory small memory leak. It can be fixed by simply add the following code to the deconstructor.

```
delete [] fileName;
```

**After further investigation it does not seem that even creating a fileName char* is even needed. The file could actually be opened with the const char* passed in through the beginning of the program. The decleartion would have to change and the function to the below.

Decleration:

```
BookParser(const char*);
```

Function:

```
BookParser::BookParser(const char* fName)
        {
            file.open(fName);
            if (!file.is_open())
                cerr << "ERROR:  Cannot open Book Parser File" <<
endl;
        }
```

```
In addition, the char* filename could be omitted from the header
file, becaues it is no longer used.
```

HelperFileParser

The helperFile is never deleted at the end of the program. This will be a small memory leak. It can be fixed by simply adding the following code to the deconstructor.

```
delete [] helperFile;
```

Word

In the function getWord, there is a char* temp created but never deleted. This could lead to a memory leak. The better way to do this would simply be to return the char* string as shown below, because this is deleted in the deconstructor.

```
char* Word::getWord()
        {
            return string;
        }
```

In the function resizePageList, there is a char* temp created but goes out of scope and is not deleted. This can be fixed by adding the code below to the end of the function.

```
delete [] temp;
```