item.def

```
<module>
<pattern>A Pattern</pattern>
<table>item</table>
<name>Item</name>
<attributes>
<attribute>
<primaryKey>true</primaryKey>
<name>code</name>
<dataType>int</dataType>
<length>4</length>
<precision>0</precision>
<required>true</required>
<column>code</column>
<autoincrement>true</autoincrement>
</attribute>
<attribute>
<name>name</name>
<searchable>true<searchable>
<dataType>int</dataType>
<unique>true</unique>
<length>25</length>
<required>true</required>
<column>name</column>
</attribute>
<attribute>
<name>salePrice</name>
<dataType>double</dataType>
<length>10</length>
<precision>2</precision>
<required>false</required>
<defaultValue>0.00</defaultValue>
<min>0.00</min>
<max>99999.99<max>
<column>sale_price</column>
</attribute>
</attributes>
</module>
```

Create classes
Attribute (write appropriate setter/getter)
properties according to the tags

Class AttributeNode
Attribute *attribute;
AttributeNode ***next,***previous
friend class AttributeDoublyLinkedList

class AttributeDoublyLinkedList
AttributeNode ***start,***end;
functions to add,insert,remove,get,getCount

Module (write appropriate setter/getter)
char name[101]
char tableName[101]
char patternName[101]
AttributeDoublyLinkedList attribute;
functions to add,remove,update,delete,get,getCount

Write a program that will read the contents of the above file

Your program should be able to read the contents of the file and parse it and a data structure should get created

---

For our parser, first of all we will have to create some tools

Logic 1 : to test StringBuffer

```
#include<stdio.h>
#include<string.h>
#include<iostream>
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
```

```
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
```

```
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};

int main()
{
/* for testing the StringBuffer, we will create a file
sb.test with multiple lines in it.
Then we will write a file handling code, which will read and create
string (one for every line) and print it
*/
FILE *f;
f=fopen("sb.test","r");
if(f==NULL)
{
printf("sb.test is missing");
return 0;
}
StringBuffer sb;
char *str=NULL;
char ch;
int lineNumber=0;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
```

```
if(str!=NULL)
{
delete [] str;
str=NULL;
}
str=sb.toString();
cout<<"Line number : "<<lineNumber<<" : "<<str<<endl;
sb.clear();
}
else
{
sb.append(ch);
}
}
fclose(f);
if(sb.characterNodeCount>0)
{
if(str!=NULL)
{
delete [] str;
str=NULL;
}
str=sb.toString();
sb.clear();
lineNumber++;
cout<<"Line number : "<<lineNumber<<" : "<<str<<endl;
delete [] str;
}
return 0;
}
```

Logic 2 (StringBufferCollection)

```
#include<stdio.h>
#include<string.h>
#include<iostream>
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
```

```
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
```

```
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
```

```
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
return t->sb;
}
```

```
};
int main()
{
/* for testing the StringBuffer, we will create a file
sb.test with multiple lines in it.
Then we will write a file handling code, which will read and create
string (one for every line) and print it
FILE *f;
f=fopen("sb.test","r");
if(f==NULL)
{
printf("sb.test is missing");
return 0;
}
StringBuffer sb;
char *str=NULL;
char ch;
int lineNumber=0;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
if(str!=NULL)
{
delete [] str;
str=NULL;
}
str=sb.toString();
cout<<"Line number : "<<lineNumber<<" : "<<str<<endl;
sb.clear();
}
else
{
sb.append(ch);
}
}
fclose(f);
if(sb.characterNodeCount>0)
{
if(str!=NULL)
{
delete [] str;
str=NULL;
```

```
}
str=sb.toString();
sb.clear();
lineNumber++;
cout<<"Line number : "<<lineNumber<<" : "<<str<<endl;
delete [] str;
}
*/


/* for testing the StringBufferCollect */
FILE *f;
f=fopen("sb.test","r");
if(f==NULL)
{
printf("sb.test is missing");
return 0;
}
StringBufferCollection sbc;
StringBuffer *sb;
char *str;
char ch;
sb=new StringBuffer;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
sbc.add(sb);
sb=new StringBuffer;
}
else
{
sb->append(ch);
}
}
fclose(f);
if(sb->characterNodeCount==0)
{
delete sb;
}
else
{
sbc.add(sb);
}
```

```
int i=0;
while(i<sbc.stringBufferNodeCount)
{
str=sbc.get(i)->toString();
cout<<"Line number : "<<(i+1)<<" : "<<str<<endl;
delete [] str;
i++;
}
return 0;
}
```

Logic (find and replace)

```
#include<stdio.h>
#include<string.h>
#include<iostream>
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
```

```
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
```

```
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
```

```
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
int endIndex=slength-flength;
while(i<=endIndex)
{
if(strncmp(s+i,f,flength)==0) return i;
i++;
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
```

```
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
strcat(s,r);
strcat(s,tmp);
}
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
int main()
{
char a[101]={"God is great. He is good."};
char b[21]={"is"};
int index;
index=indexOf(a,b);
cout<<index<<endl;
index=indexOf(a,b,index+1);
cout<<index<<endl;
int count=countOccurrences(a,b);
cout<<"Count : "<<count<<endl;
char s[1001]={"I live in Ujjain, Ujjain is a cool place, Ujjain is the city of gods"};
char f[21]={"Ujjain"};
char r[21]={"Ahmedabad"};
```

```
findAndReplace(s,f,r);
cout<<s<<endl;
return 0;
}
```

Logic of splits

```
#include<stdio.h>
#include<string.h>
#include<iostream>
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
```

```
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
```

```
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
```

```
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
int endIndex=slength-flength;
while(i<=endIndex)
{
if(strncmp(s+i,f,flength)==0) return i;
i++;
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
```

```
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
strcat(s,r);
strcat(s,tmp);
}
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
char **split(char *str,char *separator,int *numberOfSplits)
{
*numberOfSplits=countOccurrences(str,separator)+1;
char **splits=new char *[*numberOfSplits];
int leftIndex,rightIndex,separatorLength;
separatorLength=strlen(separator);
int strLength=strlen(str);
int x;
leftIndex=0;
int req;
int len;
x=0;
while(1)
{
rightIndex=indexOf(str,separator,leftIndex);
if(rightIndex==-1) break;
req=rightIndex-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strncpy(splits[x],str+leftIndex,len);
```

```
splits[x][len]='\0';
cout<<"**********************"<<endl;
cout<<splits[x]<<endl;
leftIndex=rightIndex+separatorLength;
x++;
}
req=strLength-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strcpy(splits[x],str+leftIndex);
int countOfNonEmptyStrings=0;
x=0;
while(x<*numberOfSplits)
{
if(splits[x][0]!='\0') countOfNonEmptyStrings++;
x++;
}
if(countOfNonEmptyStrings<*numberOfSplits)
{
int e,f;
char **tsplits=new char *[countOfNonEmptyStrings];
for(e=0,f=0;e<*numberOfSplits;e++)
{
if(splits[e][0]!='\0')
{
tsplits[f]=splits[e];
f++;
}
else
{
delete [] splits[e];
}
}
delete [] splits;
splits=tsplits;
}
*numberOfSplits=countOfNonEmptyStrings;
return splits;
}
int main()
{
char str[101]={"abOneabTwoabThreeababxFourabFiveabSixabSevenab"};
char separator[21]={"ab"};
int numberOfSplits;
char **splits;
splits=split(str,separator,&numberOfSplits);
int i;
```

```
cout<<"Number of splits : "<<numberOfSplits<<endl;
i=0;
while(i<numberOfSplits)
{
cout<<splits[i]<<endl;
i++;
}
i=0;
while(i<numberOfSplits)
{
delete [] splits[i];
i++;
}
delete [] splits;
return 0;
}
```

Date : 10/5/2016
test.txt (sample data file)

```
<aaa>
text1
<bbb>
bad
ugly
<ddd>
whatever1
</ddd>
whatever 2
</bbb>
text2
<ccc>
good
</ccc>
whatever 3
</aaa>
```

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#define false 0
#define true 1
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
```

```
{
next=NULL;
}
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
```

```
{
char *s;
if(start==NULL)
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
```

```
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
```

```
t=t->next;
i++;
}
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
int endIndex=slength-flength;
while(i<=endIndex)
{
if(strncmp(s+i,f,flength)==0) return i;
i++;
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
```

```
strcat(s,r);
strcat(s,tmp);
}
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
char **split(char *str,char *separator,int *numberOfSplits)
{
*numberOfSplits=countOccurrences(str,separator)+1;
char **splits=new char *[*numberOfSplits];
int leftIndex,rightIndex,separatorLength;
separatorLength=strlen(separator);
int strLength=strlen(str);
int x;
leftIndex=0;
int req;
int len;
x=0;
while(1)
{
rightIndex=indexOf(str,separator,leftIndex);
if(rightIndex==-1) break;
req=rightIndex-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strncpy(splits[x],str+leftIndex,len);
splits[x][len]='\0';
cout<<"**********************"<<endl;
cout<<splits[x]<<endl;
leftIndex=rightIndex+separatorLength;
x++;
}
req=strLength-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strcpy(splits[x],str+leftIndex);
int countOfNonEmptyStrings=0;
x=0;
while(x<*numberOfSplits)
{
if(splits[x][0]!='\0') countOfNonEmptyStrings++;
x++;
```

```
}
if(countOfNonEmptyStrings<*numberOfSplits)
{
int e,f;
char **tsplits=new char *[countOfNonEmptyStrings];
for(e=0,f=0;e<*numberOfSplits;e++)
{
if(splits[e][0]!='\0')
{
tsplits[f]=splits[e];
f++;
}
else
{
delete [] splits[e];
}
}
delete [] splits;
splits=tsplits;
}
*numberOfSplits=countOfNonEmptyStrings;
return splits;
}
class TagNode
{
public:
char *content;
int isTag;
TagNode *parent;
TagNode *next,*previous;
TagNode *start;
TagNode *end;
int childCount;
TagNode()
{
parent=start=end=NULL;
previous=next=NULL;
childCount=0;
content=NULL;
}
~TagNode()
{
TagNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
```

```
delete t;
}
end=NULL;
childCount=0;
if(content!=NULL)
{
delete [] content;
}
}
};
TagNode *start=NULL;
TagNode *end=NULL;
TagNode *current=NULL;
int isEndTag(char *str)
{
return str[1]=='/';
}
char * getEndTag(char *startTag)
{
char *str=new char[strlen(startTag)+1];
str[0]='<';
str[1]='/';
str[2]='\0';
strcat(str,startTag+1);
return str;
}
int parseFile(char *fileName)
{
FILE *f;
f=fopen(fileName,"r");
if(f==NULL)
{
cout<<fileName<<" does not exist"<<endl;
return false;
}
char ch;
int lineNumber;
int characterNumber;
lineNumber=1;
characterNumber=0;
int isPartOfTag=0;
StringBuffer sb;
char *str;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
```

```
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
characterNumber=0;
continue;
}
cout<<"("<<ch<<")";
if(ch=='<')
{
isPartOfTag=1;
if(sb.characterNodeCount>0)
{
TagNode *t;
t=new TagNode;
t->content=sb.toString();
t->parent=current;
sb.clear();
t->isTag=false;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<current->content<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
sb.append('<');
continue;
}
if(ch=='>')
```

```
{
isPartOfTag=0;
sb.append('>');
str=sb.toString();
sb.clear();
if(isEndTag(str))
{
delete [] str;
current=current->parent;
}
else
{
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
```

```
current=t;
}
continue;
}
sb.append(ch);
}
}
void traverseDataStructure()
{
cout<<"******Traversing the data structure*******"<<endl;
int numberOfTabs=0;
char *str;
int x;
TagNode *t,*j;
t=start;
while(t!=NULL)
{
x=1;
while(x<=numberOfTabs)
{
cout<<"\t";
x++;
}
cout<<t->content<<endl;
if(t->isTag)
{
numberOfTabs++;
t=t->start;
}
else
{
j=t;
t=t->next;
if(t==NULL)
{
t=j->parent;
numberOfTabs--;
if(t!=NULL)
{
str=getEndTag(t->content);
x=1;
while(x<=numberOfTabs)
{
cout<<"\t";
x++;
}
cout<<str<<endl;
```

```
delete [] str;
t=t->next;
}
}
}
}
}
void clearDataStructure()
{
if(start!=NULL)
{
delete start;
start=end=current=NULL;
}
}
int main()
{
int fileParsed=parseFile((char *)"test.txt");
if(fileParsed)
{
cout<<"File parsed"<<endl;
traverseDataStructure();
clearDataStructure();
}
else
{
cout<<"Cannot parse file"<<endl;
}
return 0;
}
```

Date : 11/5/2016

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#define false 0
#define true 1
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
```

```
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
```

```
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
```

```
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
```

```
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
int endIndex=slength-flength;
while(i<=endIndex)
{
if(strncmp(s+i,f,flength)==0) return i;
i++;
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
strcat(s,r);
strcat(s,tmp);
}
```

```
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
char **split(char *str,char *separator,int *numberOfSplits)
{
*numberOfSplits=countOccurrences(str,separator)+1;
char **splits=new char *[*numberOfSplits];
int leftIndex,rightIndex,separatorLength;
separatorLength=strlen(separator);
int strLength=strlen(str);
int x;
leftIndex=0;
int req;
int len;
x=0;
while(1)
{
rightIndex=indexOf(str,separator,leftIndex);
if(rightIndex==-1) break;
req=rightIndex-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strncpy(splits[x],str+leftIndex,len);
splits[x][len]='\0';
cout<<"**********"<<endl;
cout<<splits[x]<<endl;
leftIndex=rightIndex+separatorLength;
x++;
}
req=strLength-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strcpy(splits[x],str+leftIndex);
int countOfNonEmptyStrings=0;
x=0;
while(x<*numberOfSplits)
{
if(splits[x][0]!='\0') countOfNonEmptyStrings++;
x++;
}
if(countOfNonEmptyStrings<*numberOfSplits)
{
```

```
int e,f;
char **tsplits=new char *[countOfNonEmptyStrings];
for(e=0,f=0;e<*numberOfSplits;e++)
{
if(splits[e][0]!='\0')
{
tsplits[f]=splits[e];
f++;
}
else
{
delete [] splits[e];
}
}
delete [] splits;
splits=tsplits;
}
*numberOfSplits=countOfNonEmptyStrings;
return splits;
}
class TagNode
{
public:
char *content;
int isTag;
TagNode *parent;
TagNode *next,*previous;
TagNode *start;
TagNode *end;
int childCount;
TagNode()
{
parent=start=end=NULL;
previous=next=NULL;
childCount=0;
content=NULL;
}
~TagNode()
{
TagNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
```

```
childCount=0;
if(content!=NULL)
{
delete [] content;
}
}
};
TagNode *start=NULL;
TagNode *end=NULL;
TagNode *current=NULL;
int isEndTag(char *str)
{
return str[1]=='/';
}
char * getEndTag(char *startTag)
{
char *str=new char[strlen(startTag)+1];
str[0]='<';
str[1]='/';
str[2]='\0';
strcat(str,startTag+1);
return str;
}
void allTrim(char *str)
{
int l;
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
}
```

```
void rightTrim(char *str)
{
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
}
void leftTrim(char *str)
{
int l;
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
}
void trimTag(char *str)
{
while(countOccurrences(str,(char *)"< ")>0)
{
findAndReplace(str,(char *)"< ",(char *)"<");
}
while(countOccurrences(str,(char *)" >")>0)
{
findAndReplace(str,(char *)" >",(char *)">");
}
if(isEndTag(str))
{
while(countOccurrences(str,(char *)"/ ")>0)
{
findAndReplace(str,(char *)"/ ",(char *)"/");
}
}
else
{
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
```

```
}
}
int parseFile(char *fileName)
{

FILE *f;
f=fopen(fileName,"r");
if(f==NULL)
{
cout<<fileName<<" does not exist"<<endl;
return false;
}
int lessThanHandled=1;
char ch;
int lineNumber;
int characterNumber;
lineNumber=1;
characterNumber=0;
int isPartOfTag=0;
StringBuffer sb;
char *str,*str2;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
characterNumber=0;
continue;
}
characterNumber++;
if(ch=='<')
{
if(!lessThanHandled)
{
cout<<"Expected >, found < at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=false;
isPartOfTag=1;
if(sb.characterNodeCount>0)
{
TagNode *t;
t=new TagNode;
t->content=sb.toString();
```

```
t->parent=current;
sb.clear();
t->isTag=false;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<current->content<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
sb.append('<');
continue;
}
if(ch=='>')
{
if(lessThanHandled)
{
cout<<"Expected <, found > at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=true;
isPartOfTag=0;
sb.append('>');
str=sb.toString();
trimTag(str);
sb.clear();
if(isEndTag(str))
{
str2=getEndTag(current->content);
if(stricmp(str,str2)!=0)
```

```
{
printf("%s, contains malformed tags, end tag for %s missing\n",fileName,current->content);
delete [] str2;
delete [] str;
return 0;
}
delete [] str2;
delete [] str;
current=current->parent;
}
else
{
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
```

```
current=t;
}
continue;
}
sb.append(ch);
}
}
void traverseDataStructure()
{
cout<<"******Traversing the data structure*******"<<endl;
int numberOfTabs=0;
char *str;
int x;
TagNode *t,*j;
t=start;
while(t!=NULL)
{
x=1;
while(x<=numberOfTabs)
{
cout<<"\t";
x++;
}
cout<<t->content<<endl;
if(t->isTag)
{
if(t->start==NULL)
{
str=getEndTag(t->content);
x=1;
while(x<=numberOfTabs)
{
cout<<"\t";
x++;
}
cout<<str<<endl;
delete [] str;
t=t->next;
}
else
{
numberOfTabs++;
t=t->start;
}
}
else
{
```

```
j=t;
t=t->next;
if(t==NULL)
{
t=j->parent;
numberOfTabs--;
if(t!=NULL)
{
str=getEndTag(t->content);
x=1;
while(x<=numberOfTabs)
{
cout<<"\t";
x++;
}
cout<<str<<endl;
delete [] str;
t=t->next;
}
}
}
}
}
void clearDataStructure()
{
if(start!=NULL)
{
delete start;
start=end=current=NULL;
}
}
int main()
{
int fileParsed=parseFile((char *)"test.txt");
if(fileParsed)
{
cout<<"File parsed"<<endl;
traverseDataStructure();
clearDataStructure();
}
else
{
cout<<"Cannot parse file"<<endl;
}
return 0;
}
```

```
<      aaa        >
text1
<bbb       >
bad
ugly
<      ddd>
whatever1
<      /   ddd>
whatever 2
</    bbb      >
text2
<iiii><jjj></jjj>
</iiii><ppp></ppp><cccs></cccs>
<    ccc      >
good
<    /     ccc     >
whatever 3
<         /aaa   >
```

```
Date : 12/5/2016
#include<stdio.h>
#include<string.h>
#include<iostream>
#define false 0
#define true 1
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
```

```
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
```

```
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
```

```
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
int endIndex=slength-flength;
while(i<=endIndex)
{
```

```
if(strncmp(s+i,f,flength)==0) return i;
i++;
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
strcat(s,r);
strcat(s,tmp);
}
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
char **split(char *str,char *separator,int *numberOfSplits)
{
*numberOfSplits=countOccurrences(str,separator)+1;
char **splits=new char *[*numberOfSplits];
```

```
int leftIndex,rightIndex,separatorLength;
separatorLength=strlen(separator);
int strLength=strlen(str);
int x;
leftIndex=0;
int req;
int len;
x=0;
while(1)
{
rightIndex=indexOf(str,separator,leftIndex);
if(rightIndex==-1) break;
req=rightIndex-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strncpy(splits[x],str+leftIndex,len);
splits[x][len]='\0';
cout<<"**********************"<<endl;
cout<<splits[x]<<endl;
leftIndex=rightIndex+separatorLength;
x++;
}
req=strLength-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strcpy(splits[x],str+leftIndex);
int countOfNonEmptyStrings=0;
x=0;
while(x<*numberOfSplits)
{
if(splits[x][0]!='\0') countOfNonEmptyStrings++;
x++;
}
if(countOfNonEmptyStrings<*numberOfSplits)
{
int e,f;
char **tsplits=new char *[countOfNonEmptyStrings];
for(e=0,f=0;e<*numberOfSplits;e++)
{
if(splits[e][0]!='\0')
{
tsplits[f]=splits[e];
f++;
}
else
{
delete [] splits[e];
```

```
}
}
delete [] splits;
splits=tsplits;
}
*numberOfSplits=countOfNonEmptyStrings;
return splits;
}
class TagNode
{
public:
char *content;
int isTag;
TagNode *parent;
TagNode *next,*previous;
TagNode *start;
TagNode *end;
int childCount;
TagNode()
{
parent=start=end=NULL;
previous=next=NULL;
childCount=0;
content=NULL;
}
~TagNode()
{
TagNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
childCount=0;
if(content!=NULL)
{
delete [] content;
}
}
};
TagNode *start=NULL;
TagNode *end=NULL;
TagNode *current=NULL;
int isSlashAtCorrectPosition(char *str)
{
```

```
int i=countOccurrences(str,(char *)"/");
if(i==0) return true;
if(i>1) return false;
if(str[1]='/') return true;
if(str[strlen(str)-2]=='/') return true;
return false;
}
int isEndTag(char *str)
{
return str[1]=='/';
}
int isStartAndEndTag(char *str)
{
return str[strlen(str)-2]=='/';
}

char * getEndTag(char *startTag)
{
char *str=new char[strlen(startTag)+1];
str[0]='<';
str[1]='/';
str[2]='\0';
strcat(str,startTag+1);
return str;
}
void allTrim(char *str)
{
int l;
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
```

```
}
}

void rightTrim(char *str)
{
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
}
void leftTrim(char *str)
{
int l;
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
}
void trimTag(char *str)
{
while(countOccurrences(str,(char *)"< ")>0)
{
findAndReplace(str,(char *)"< ",(char *)"<");
}
while(countOccurrences(str,(char *)" >")>0)
{
findAndReplace(str,(char *)" >",(char *)">");
}
while(countOccurrences(str,(char *)" /")>0)
{
findAndReplace(str,(char *)" /",(char *)"/");
}
if(isEndTag(str))
{
while(countOccurrences(str,(char *)"/ ")>0)
{
findAndReplace(str,(char *)"/ ",(char *)"/");
}
}
```

```
else
{
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
}
}
int parseFile(char *fileName)
{
int i;
FILE *f;
f=fopen(fileName,"r");
if(f==NULL)
{
cout<<fileName<<" does not exist"<<endl;
return false;
}
int lessThanHandled=1;
char ch;
int lineNumber;
int characterNumber;
lineNumber=1;
characterNumber=0;
int isPartOfTag=0;
StringBuffer sb;
char *str,*str2;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
characterNumber=0;
continue;
}
characterNumber++;
if(ch=='<')
{
if(!lessThanHandled)
{
cout<<"Expected >, found < at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=false;
```

```
isPartOfTag=1;
if(sb.characterNodeCount>0)
{
TagNode *t;
t=new TagNode;
t->content=sb.toString();
t->parent=current;
sb.clear();
t->isTag=false;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<current->content<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
sb.append('<');
continue;
}
if(ch=='>')
{
if(lessThanHandled)
{
cout<<"Expected <, found > at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=true;
isPartOfTag=0;
sb.append('>');
str=sb.toString();
```

```
trimTag(str);
sb.clear();
if(!isSlashAtCorrectPosition(str))
{
cout<<"Incorrect format of tag "<<str<<" at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
if(isStartAndEndTag(str))
{
i=strlen(str);
str[i-2]='>';
str[i-1]='\0';
i--;
if(i==2)
{
cout<<"Invalid start tag <> at "<<lineNumber<<","<<characterNumber-1<<endl;
delete [] str;
return false;
}
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
```

```
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
// current=t;   // because an empty tag got created
continue;
}
if(isEndTag(str))
{
if(strlen(str)==3)
{
cout<<"Invalid end tag </> at "<<lineNumber<<","<<characterNumber-2<<endl;
delete [] str;
return false;
}
str2=getEndTag(current->content);
if(stricmp(str,str2)!=0)
{
printf("%s, contains malformed tags, end tag for %s missing\n",fileName,current->content);
delete [] str2;
delete [] str;
return 0;
}
delete [] str2;
delete [] str;
current=current->parent;
}
else
{
if(strlen(str)==2)
{
cout<<"Invalid start tag <> at "<<lineNumber<<","<<characterNumber-1<<endl;
delete [] str;
return false;
}
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
```

```
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
current=t;
}
continue;
}
sb.append(ch);
}
}
void traverseDataStructure()
{
cout<<"******Traversing the data structure*******"<<endl;
int numberOfTabs=0;
char *str;
int x;
TagNode *t,*j;
j=NULL;
t=start;
while(1)
{
if(t==NULL)
{
if(j==NULL) break;
```

```
str=getEndTag(j->content);
cout<<str<<endl;
delete [] str;
t=j;
j=j->parent;
t=t->next;
if(t==NULL) continue;
}
cout<<t->content<<endl;
if(t->isTag)
{
j=t;
t=t->start;
}
else
{
t=t->next;
}
}
}
void clearDataStructure()
{
if(start!=NULL)
{
delete start;
start=end=current=NULL;
}
}
int main()
{
int fileParsed=parseFile((char *)"test.txt");
if(fileParsed)
{
cout<<"File parsed"<<endl;
traverseDataStructure();
clearDataStructure();
}
else
{
cout<<"Cannot parse file"<<endl;
}
return 0;
}
```

---

test.txt

```
<     aaa       >
text1
```

<bbb >
bad
ugly
< ddd / >
whatever1
whatever 2
</ bbb >
text2
<iiii><jjj></jjj>
</iiii><ppp></ppp><cccs></cccs>
< ccc >
good
< / ccc >
whatever 3
< /aaa >

Date : 13/5/2016
tools.cpp

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#define false 0
#define true 1
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
characterNodeCount=0;
}
~StringBuffer()
{
clear();
```

```
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
```

```
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
```

```
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
int endIndex=slength-flength;
while(i<=endIndex)
{
if(strncmp(s+i,f,flength)==0) return i;
i++;
```

```
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
strcat(s,r);
strcat(s,tmp);
}
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
char **split(char *str,char *separator,int *numberOfSplits)
{
*numberOfSplits=countOccurrences(str,separator)+1;
char **splits=new char *[*numberOfSplits];
int leftIndex,rightIndex,separatorLength;
separatorLength=strlen(separator);
```

```
int strLength=strlen(str);
int x;
leftIndex=0;
int req;
int len;
x=0;
while(1)
{
rightIndex=indexOf(str,separator,leftIndex);
if(rightIndex==-1) break;
req=rightIndex-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strncpy(splits[x],str+leftIndex,len);
splits[x][len]='\0';
cout<<"**********"<<endl;
cout<<splits[x]<<endl;
leftIndex=rightIndex+separatorLength;
x++;
}
req=strLength-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strcpy(splits[x],str+leftIndex);
int countOfNonEmptyStrings=0;
x=0;
while(x<*numberOfSplits)
{
if(splits[x][0]!='\0') countOfNonEmptyStrings++;
x++;
}
if(countOfNonEmptyStrings<*numberOfSplits)
{
int e,f;
char **tsplits=new char *[countOfNonEmptyStrings];
for(e=0,f=0;e<*numberOfSplits;e++)
{
if(splits[e][0]!='\0')
{
tsplits[f]=splits[e];
f++;
}
else
{
delete [] splits[e];
}
}
```

```
delete [] splits;
splits=tsplits;
}
*numberOfSplits=countOfNonEmptyStrings;
return splits;
}
class AttributeNode
{
public:
char *name;
char *value;
AttributeNode *next,*previous;
AttributeNode()
{
next=previous=NULL;
name=value=NULL;
}
~AttributeNode()
{
if(name!=NULL) delete [] name;
name=NULL;
if(value!=NULL) delete [] value;
value=NULL;
}
};
class TagNode
{
public:
char *content;
int isTag;
TagNode *parent;
TagNode *next,*previous;
TagNode *start;
TagNode *end;
int childCount;
AttributeNode *attributeStart,*attributeEnd;
int attributesCount;
TagNode()
{
parent=start=end=NULL;
previous=next=NULL;
childCount=0;
content=NULL;
attributeStart=attributeEnd=NULL;
attributesCount=0;
}
~TagNode()
```

```
{
TagNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
childCount=0;
if(content!=NULL)
{
delete [] content;
}
AttributeNode *at;
while(attributeStart!=NULL)
{
at=attributeStart;
attributeStart=attributeStart->next;
delete at;
}
attributeEnd=NULL;
attributesCount=0;
}
};
TagNode *start=NULL;
TagNode *end=NULL;
TagNode *current=NULL;
int isSlashAtCorrectPosition(char *str)
{
int i=countOccurrences(str,(char *)"/");
if(i==0) return true;
if(i>1) return false;
if(str[1]=='/') return true;
if(str[strlen(str)-2]=='/') return true;
return false;
}
int isEndTag(char *str)
{
return str[1]=='/';
}
int isStartAndEndTag(char *str)
{
return str[strlen(str)-2]=='/';
}

char * getEndTag(char *startTag)
```

```
{
char *str=new char[strlen(startTag)+1];
str[0]='<';
str[1]='/';
str[2]='\0';
strcat(str,startTag+1);
return str;
}
void allTrim(char *str)
{
int l;
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
}

void rightTrim(char *str)
{
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
}
void leftTrim(char *str)
{
int l;
l=0;
```

```
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
}
void trimTag(char *str)
{
while(countOccurrences(str,(char *)"< ")>0)
{
findAndReplace(str,(char *)"< ",(char *)"<");
}
while(countOccurrences(str,(char *)" >")>0)
{
findAndReplace(str,(char *)" >",(char *)">");
}
while(countOccurrences(str,(char *)" /")>0)
{
findAndReplace(str,(char *)" /",(char *)"/");
}
if(isEndTag(str))
{
while(countOccurrences(str,(char *)"/ ")>0)
{
findAndReplace(str,(char *)"/ ",(char *)"/");
}
}
else
{
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
}
}
int parseFile(char *fileName)
{
AttributeNode *at,*astart,*aend;
char **nv;
char **attributes;
int m,u;
int i;
FILE *f;
f=fopen(fileName,"r");
```

```
if(f==NULL)
{
cout<<fileName<<" does not exist"<<endl;
return false;
}
int lessThanHandled=1;
char ch;
int numberOfSplits;
int lineNumber;
int characterNumber;
lineNumber=1;
characterNumber=0;
int isPartOfTag=0;
StringBuffer sb;
char *str,*str2;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
characterNumber=0;
continue;
}
characterNumber++;
if(ch=='<')
{
if(!lessThanHandled)
{
cout<<"Expected >, found < at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=false;
isPartOfTag=1;
if(sb.characterNodeCount>0)
{
TagNode *t;
t=new TagNode;
t->content=sb.toString();
t->parent=current;
sb.clear();
t->isTag=false;
// logic to debug starts
if(current!=NULL)
{
```

```
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<current->content<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
sb.append('<');
continue;
}
if(ch=='>')
{
if(lessThanHandled)
{
cout<<"Expected <, found > at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=true;
isPartOfTag=0;
sb.append('>');
str=sb.toString();
trimTag(str);
sb.clear();
if(!isSlashAtCorrectPosition(str))
{
cout<<"Incorrect format of tag "<<str<<" at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
if(isStartAndEndTag(str))
{
i=strlen(str);
str[i-2]='>';
str[i-1]='\0';
```

```
i--;
if(i==2)
{
cout<<"Invalid start tag <> at "<<lineNumber<<","<<characterNumber-1<<endl;
delete [] str;
return false;
}
astart=aend=NULL;
numberOfSplits=0;
if(countOccurrences(str,(char *)" ")>0)
{
// contains attributes
attributes=split(str,(char *)" ",&numberOfSplits);
//cout<<numberOfSplits<<endl;
strcpy(str,attributes[0]);
m=strlen(str);
str[m]='>';
str[m+1]='\0';
//cout<<str<<endl;
m=strlen(attributes[numberOfSplits-1]);
if(attributes[numberOfSplits-1][m-2]=='/')
{
attributes[numberOfSplits-1][m-2]='\0';
}
else
{
attributes[numberOfSplits-1][m-1]='\0';
}
m=1;
while(m<numberOfSplits)
{
nv=split(attributes[m],(char *)"=",&u);
// need to apply validations
at=new AttributeNode;
at->name=nv[0];
at->value=new char[strlen(nv[1])-2+1];
strcpy(at->value,nv[1]+1);
at->value[strlen(at->value)-1]='\0';
delete [] attributes[m];
delete [] nv[1];
if(astart==NULL)
{
astart=aend=at;
}
else
{
aend->next=at;
```

```
at->previous=aend;
aend=at;
}
m++;
}
}
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
t->attributeStart=astart;
t->attributeEnd=aend;
t->attributesCount=numberOfSplits-1;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
// current=t;   // because an empty tag got created
continue;
}
```

```
if(isEndTag(str))
{
if(strlen(str)==3)
{
cout<<"Invalid end tag </> at "<<lineNumber<<","<<characterNumber-2<<endl;
delete [] str;
return false;
}
str2=getEndTag(current->content);
if(stricmp(str,str2)!=0)
{
printf("%s, contains malformed tags, end tag for %s missing\n",fileName,current->content);
delete [] str2;
delete [] str;
return 0;
}
delete [] str2;
delete [] str;
current=current->parent;
}
else
{
if(strlen(str)==2)
{
cout<<"Invalid start tag <> at "<<lineNumber<<","<<characterNumber-1<<endl;
delete [] str;
return false;
}
astart=aend=NULL;
numberOfSplits=0;
if(countOccurrences(str,(char *)" ")>0)
{
// contains attributes
attributes=split(str,(char *)" ",&numberOfSplits);
//cout<<numberOfSplits<<endl;
strcpy(str,attributes[0]);
m=strlen(str);
str[m]='>';
str[m+1]='\0';
//cout<<str<<endl;
m=strlen(attributes[numberOfSplits-1]);
if(attributes[numberOfSplits-1][m-2]=='/')
{
attributes[numberOfSplits-1][m-2]='\0';
}
else
{
```

```
attributes[numberOfSplits-1][m-1]='\0';
}
m=1;
while(m<numberOfSplits)
{
nv=split(attributes[m],(char *)"=",&u);
// need to apply validations
at=new AttributeNode;
at->name=nv[0];
at->value=new char[strlen(nv[1])-2+1];
strcpy(at->value,nv[1]+1);
at->value[strlen(at->value)-1]='\0';
delete [] attributes[m];
delete [] nv[1];
if(astart==NULL)
{
astart=aend=at;
}
else
{
aend->next=at;
at->previous=aend;
aend=at;
}
m++;
}
}
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
t->attributeStart=astart;
t->attributeEnd=aend;
t->attributesCount=numberOfSplits-1;
// logic to debug starts
if(current!=NULL)
{
cout<<"Append under "<<current->content<<endl;
cout<<t->content<<endl;
}
else
{
cout<<"root element"<<endl;
cout<<t->content<<endl;
}
//logic to debug ends
```

```
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
current=t;
}
continue;
}
sb.append(ch);
}
}
void traverseDataStructure()
{
cout<<"******Traversing the data structure*******"<<endl;
int numberOfTabs=0;
char *str;
int x;
TagNode *t,*j;
AttributeNode *at;
j=NULL;
t=start;
while(1)
{
if(t==NULL)
{
if(j==NULL) break;
str=getEndTag(j->content);
cout<<str<<endl;
delete [] str;
t=j;
j=j->parent;
t=t->next;
if(t==NULL) continue;
```

```
}
cout<<t->content<<endl;
if(t->isTag)
{
at=t->attributeStart;
if(at!=NULL)
{
cout<<"Number of attribute : "<<t->attributesCount<<endl;
}
while(at!=NULL)
{
cout<<"("<<at->name<<","<<at->value<<")  ";
at=at->next;
if(at==NULL)
{
cout<<endl;
}
}
j=t;
t=t->start;
}
else
{
t=t->next;
}
}
}
void clearDataStructure()
{
if(start!=NULL)
{
delete start;
start=end=current=NULL;
}
}
int main()
{
int fileParsed=parseFile((char *)"test.txt");
if(fileParsed)
{
cout<<"File parsed"<<endl;
traverseDataStructure();
clearDataStructure();
}
else
{
cout<<"Cannot parse file"<<endl;
```

```
}
return 0;
}
```

---

test.txt

```
<     aaa    module='CoolStuff'    version="1.0"    >
text1
<bbb      good='bad'   name='cool' >
bad
ugly
<      ddd  font='Arial' size='24'  /   >
whatever1
whatever 2
</    bbb      >
text2
<iiii><jjj name='Gopal'></jjj>
</iiii><ppp></ppp><cccs></cccs>
<    ccc      >
good
<    /     ccc     >
whatever 3
<        /aaa   >
```

Date : 16/5/2016

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#define false 0
#define true 1
using namespace std;
class CharacterNode
{
public:
char c;
CharacterNode *next;
CharacterNode()
{
next=NULL;
}
};
class StringBuffer
{
public:
CharacterNode *start,*end;
int characterNodeCount;
StringBuffer()
{
start=end=NULL;
```

```
characterNodeCount=0;
}
~StringBuffer()
{
clear();
}
void append(char c)
{
CharacterNode *t;
t=new CharacterNode;
t->c=c;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
end=t;
}
characterNodeCount++;
}
void clear()
{
CharacterNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
characterNodeCount=0;
}
char * toString()
{
char *s;
if(start==NULL)
{
s=new char[1];
s[0]='\0';
}
else
{
s=new char[characterNodeCount+1];
int i;
i=0;
```

```
CharacterNode *t;
t=start;
while(t!=NULL)
{
s[i]=t->c;
t=t->next;
i++;
}
s[i]='\0';
}
return s;
}
};
class StringBufferNode
{
public:
StringBuffer *sb;
StringBufferNode *next,*previous;
StringBufferNode()
{
next=previous=NULL;
sb=NULL;
}
~StringBufferNode()
{
delete sb;
}
};
class StringBufferCollection
{
public:
StringBufferNode *start,*end;
int stringBufferNodeCount;
StringBufferCollection()
{
start=end=NULL;
stringBufferNodeCount=0;
}
~StringBufferCollection()
{
clear();
}
void clear()
{
StringBufferNode *t;
while(start!=NULL)
{
```

```
t=start;
start=start->next;
delete t;
}
end=NULL;
stringBufferNodeCount=0;
}
void add(StringBuffer *sb)
{
StringBufferNode *t;
t=new StringBufferNode;
t->sb=sb;
if(start==NULL)
{
start=end=t;
}
else
{
end->next=t;
t->previous=end;
end=t;
}
stringBufferNodeCount++;
}
StringBuffer * get(int index)
{
if(index<0 || index>=stringBufferNodeCount)
{
return NULL;
}
StringBufferNode *t;
int i=0;
t=start;
while(i<index)
{
t=t->next;
i++;
}
return t->sb;
}
};
int indexOf(char *s,char *f,int startFromIndex=0)
{
if(startFromIndex<0) return -1;
int i=0+startFromIndex;
int slength=strlen(s);
int flength=strlen(f);
```

```
int endIndex=slength-flength;
while(i<=endIndex)
{
if(strncmp(s+i,f,flength)==0) return i;
i++;
}
return -1;
}
int countOccurrences(char *s,char *f)
{
int count=0;
int startFromIndex=0;
while(1)
{
startFromIndex=indexOf(s,f,startFromIndex);
if(startFromIndex==-1) break;
count++;
startFromIndex++;
}
return count;
}
void findAndReplace(char *s,char *f,char *r)
{
int si=0;
int flength=strlen(f);
int rlength=strlen(r);
char *tmp=new char[strlen(s)+1];
while(1)
{
si=indexOf(s,f,si);
if(si==-1) break;
if(flength!=rlength)
{
strcpy(tmp,s+si+flength);
s[si]='\0';
strcat(s,r);
strcat(s,tmp);
}
else
{
strncpy(s+si,r,rlength); // strncpy does not place \0
}
si=si+rlength;
}
delete [] tmp;
}
char **split(char *str,char *separator,int *numberOfSplits)
```

```
{
*numberOfSplits=countOccurrences(str,separator)+1;
char **splits=new char *[*numberOfSplits];
int leftIndex,rightIndex,separatorLength;
separatorLength=strlen(separator);
int strLength=strlen(str);
int x;
leftIndex=0;
int req;
int len;
x=0;
while(1)
{
rightIndex=indexOf(str,separator,leftIndex);
if(rightIndex==-1) break;
req=rightIndex-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strncpy(splits[x],str+leftIndex,len);
splits[x][len]='\0';
leftIndex=rightIndex+separatorLength;
x++;
}
req=strLength-leftIndex+1;
len=req-1;
splits[x]=new char[req];
strcpy(splits[x],str+leftIndex);
int countOfNonEmptyStrings=0;
x=0;
while(x<*numberOfSplits)
{
if(splits[x][0]!='\0') countOfNonEmptyStrings++;
x++;
}
if(countOfNonEmptyStrings<*numberOfSplits)
{
int e,f;
char **tsplits=new char *[countOfNonEmptyStrings];
for(e=0,f=0;e<*numberOfSplits;e++)
{
if(splits[e][0]!='\0')
{
tsplits[f]=splits[e];
f++;
}
else
{
```

```
delete [] splits[e];
}
}
delete [] splits;
splits=tsplits;
}
*numberOfSplits=countOfNonEmptyStrings;
return splits;
}
class AttributeNode
{
public:
char *name;
char *value;
AttributeNode *next,*previous;
AttributeNode()
{
next=previous=NULL;
name=value=NULL;
}
~AttributeNode()
{
if(name!=NULL) delete [] name;
name=NULL;
if(value!=NULL) delete [] value;
value=NULL;
}
};
class TagNode
{
public:
char *content;
int isTag;
TagNode *parent;
TagNode *next,*previous;
TagNode *start;
TagNode *end;
int childCount;
AttributeNode *attributeStart,*attributeEnd;
int attributesCount;
TagNode()
{
parent=start=end=NULL;
previous=next=NULL;
childCount=0;
content=NULL;
attributeStart=attributeEnd=NULL;
```

```
attributesCount=0;
}
~TagNode()
{
TagNode *t;
while(start!=NULL)
{
t=start;
start=start->next;
delete t;
}
end=NULL;
childCount=0;
if(content!=NULL)
{
delete [] content;
}
AttributeNode *at;
while(attributeStart!=NULL)
{
at=attributeStart;
attributeStart=attributeStart->next;
delete at;
}
attributeEnd=NULL;
attributesCount=0;
}
};
TagNode *start=NULL;
TagNode *end=NULL;
TagNode *current=NULL;
int isSlashAtCorrectPosition(char *str)
{
int i=countOccurrences(str,(char *)"/");
if(i==0) return true;
if(i>1) return false;
if(str[1]=='/') return true;
if(str[strlen(str)-2]=='/') return true;
return false;
}
int isEndTag(char *str)
{
return str[1]=='/';
}
int isStartAndEndTag(char *str)
{
return str[strlen(str)-2]=='/';
```

```
}

char * getEndTag(char *startTag)
{
char *str=new char[strlen(startTag)+1];
str[0]='<';
str[1]='/';
str[2]='\0';
strcat(str,startTag+1);
return str;
}
void allTrim(char *str)
{
int l;
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
}

void rightTrim(char *str)
{
int r;
r=strlen(str)-1;
while(str[r]==' ' && r>=0)
{
r--;
}
str[r+1]='\0';
}
void leftTrim(char *str)
```

```
{
int l;
l=0;
while(str[l]==' ')
{
l++;
}
if(l>0)
{
strcpy(str,str+l);
}
}
void trimTag(char *str)
{
while(countOccurrences(str,(char *)"< ")>0)
{
findAndReplace(str,(char *)"< ",(char *)"<");
}
while(countOccurrences(str,(char *)" >")>0)
{
findAndReplace(str,(char *)" >",(char *)">");
}
while(countOccurrences(str,(char *)" /")>0)
{
findAndReplace(str,(char *)" /",(char *)"/");
}
if(isEndTag(str))
{
while(countOccurrences(str,(char *)"/ ")>0)
{
findAndReplace(str,(char *)"/ ",(char *)"/");
}
}
else
{
while(countOccurrences(str,(char *)"  ")>0)
{
findAndReplace(str,(char *)"  ",(char *)" ");
}
}
}
char ** splitNameValuePair(char *str)
{
/*48-57 but not at index 0
65 - 90 anywhere
97 - 122 anywhere
36 for $ anywhere
```

```
95 for _ anywhere*/
int x,y,z;
x=0;
while(str[x]!='\0' && str[x]!='=')
{
x++;
}
if(str[x]=='\0')
{
return NULL;
}
if(str[x+1]=='\0')
{
return NULL;
}
char **nv=new char *[2];
nv[0]=new char[x+1];
nv[1]=new char[strlen(str)-x+1];
strncpy(nv[0],str,x);
nv[0][x]='\0';
strcpy(nv[1],str+x+1);
allTrim(nv[0]);
x=strlen(nv[0]);
y=0;
while(y<x)
{
if(!((nv[0][y]>=65 && nv[0][y]<=90) || (nv[0][y]>=97 && nv[0][y]<=122) || nv[0][y]==36 || nv[0]
[y]==95 || (x>0 && nv[0][y]>=48 && nv[0][y]<=57)))
{
delete [] nv[0];
delete [] nv[1];
delete [] nv;
return NULL;
}
y++;
}
allTrim(nv[1]);
x=strlen(nv[1]);
// 39 for ' and 34 for "
if(nv[1][0]!=39  && nv[1][0]!=34)
{
delete [] nv[0];
delete [] nv[1];
delete [] nv;
return NULL;
}
if(nv[1][x-1]!=39  && nv[1][x-1]!=34)
```

```
{
delete [] nv[0];
delete [] nv[1];
delete [] nv;
return NULL;
}
if(nv[1][0]!=nv[1][x-1])
{
delete [] nv[0];
delete [] nv[1];
delete [] nv;
return NULL;
}
int na=(nv[1][0]==39)?39:34;
y=1;
while(y<x-1)
{
if(nv[1][y]==na)
{
delete [] nv[0];
delete [] nv[1];
delete [] nv;
return NULL;
}
y++;
}
char *name=new char[strlen(nv[0])+1];
char *value=new char[strlen(nv[1])+1];
strcpy(name,nv[0]);
strcpy(value,nv[1]);
delete [] nv[0];
delete [] nv[1];
nv[0]=name;
nv[1]=value;
return nv;
}
int parseFile(char *fileName)
{
AttributeNode *at,*astart,*aend;
char **nv;
char **attributes;
int m,u;
int i;
FILE *f;
f=fopen(fileName,"r");
if(f==NULL)
{
```

```
cout<<fileName<<" does not exist"<<endl;
return false;
}
int lessThanHandled=1;
char ch;
int numberOfSplits;
int lineNumber;
int characterNumber;
lineNumber=1;
characterNumber=0;
int isPartOfTag=0;
StringBuffer sb;
char *str,*str2;
while(1)
{
ch=fgetc(f);
if(feof(f)) break;
if(ch=='\r') continue;
if(ch=='\n')
{
lineNumber++;
characterNumber=0;
continue;
}
characterNumber++;
if(ch=='<')
{
if(!lessThanHandled)
{
cout<<"Expected >, found < at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=false;
isPartOfTag=1;
if(sb.characterNodeCount>0)
{
TagNode *t;
t=new TagNode;
t->content=sb.toString();
t->parent=current;
sb.clear();
t->isTag=false;
// logic to debug starts
if(current!=NULL)
{
//cout<<"Append under "<<current->content<<endl;
//cout<<t->content<<endl;
```

```
}
else
{
//cout<<"root element"<<current->content<<endl;
//cout<<t->content<<endl;
}
//logic to debug ends
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
sb.append('<');
continue;
}
if(ch=='>')
{
if(lessThanHandled)
{
cout<<"Expected <, found > at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
lessThanHandled=true;
isPartOfTag=0;
sb.append('>');
str=sb.toString();
trimTag(str);
sb.clear();
if(!isSlashAtCorrectPosition(str))
{
cout<<"Incorrect format of tag "<<str<<" at "<<lineNumber<<","<<characterNumber<<endl;
return false;
}
if(isStartAndEndTag(str))
{
i=strlen(str);
str[i-2]='>';
str[i-1]='\0';
i--;
if(i==2)
```

```
{
cout<<"Invalid start tag <> at "<<lineNumber<<","<<characterNumber-1<<endl;
delete [] str;
return false;
}
astart=aend=NULL;
numberOfSplits=0;
if(countOccurrences(str,(char *)" ")>0)
{
// contains attributes
attributes=split(str,(char *)" ",&numberOfSplits);
//cout<<numberOfSplits<<endl;
strcpy(str,attributes[0]);
m=strlen(str);
str[m]='>';
str[m+1]='\0';
//cout<<str<<endl;
m=strlen(attributes[numberOfSplits-1]);
if(attributes[numberOfSplits-1][m-2]=='/')
{
attributes[numberOfSplits-1][m-2]='\0';
}
else
{
attributes[numberOfSplits-1][m-1]='\0';
}
m=1;
while(m<numberOfSplits)
{
nv=splitNameValuePair((char *)attributes[m]);
if(nv==NULL)
{
cout<<"Attributes of "<<attributes[0]<<"> are invalid"<<endl;
return false;
}
at=new AttributeNode;
at->name=nv[0];
at->value=new char[strlen(nv[1])-2+1];
strcpy(at->value,nv[1]+1);
at->value[strlen(at->value)-1]='\0';
delete [] attributes[m];
delete [] nv[1];
if(astart==NULL)
{
astart=aend=at;
}
else
```

```
{
aend->next=at;
at->previous=aend;
aend=at;
}
m++;
}
}
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
t->attributeStart=astart;
t->attributeEnd=aend;
t->attributesCount=numberOfSplits-1;
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
// current=t;   // because an empty tag got created
continue;
}
if(isEndTag(str))
{
if(strlen(str)==3)
{
cout<<"Invalid end tag </> at "<<lineNumber<<","<<characterNumber-2<<endl;
delete [] str;
return false;
}
str2=getEndTag(current->content);
if(stricmp(str,str2)!=0)
```

```
{
printf("%s, contains malformed tags, end tag for %s missing\n",fileName,current->content);
delete [] str2;
delete [] str;
return 0;
}
delete [] str2;
delete [] str;
current=current->parent;
}
else
{
if(strlen(str)==2)
{
cout<<"Invalid start tag <> at "<<lineNumber<<","<<characterNumber-1<<endl;
delete [] str;
return false;
}
astart=aend=NULL;
numberOfSplits=0;
if(countOccurrences(str,(char *)" ")>0)
{
// contains attributes
attributes=split(str,(char *)" ",&numberOfSplits);
//cout<<numberOfSplits<<endl;
strcpy(str,attributes[0]);
m=strlen(str);
str[m]='>';
str[m+1]='\0';
//cout<<str<<endl;
m=strlen(attributes[numberOfSplits-1]);
if(attributes[numberOfSplits-1][m-2]=='/')
{
attributes[numberOfSplits-1][m-2]='\0';
}
else
{
attributes[numberOfSplits-1][m-1]='\0';
}
m=1;
while(m<numberOfSplits)
{
nv=split(attributes[m],(char *)"=",&u);
// need to apply validations
at=new AttributeNode;
at->name=nv[0];
at->value=new char[strlen(nv[1])-2+1];
```

```
strcpy(at->value,nv[1]+1);
at->value[strlen(at->value)-1]='\0';
delete [] attributes[m];
delete [] nv[1];
if(astart==NULL)
{
astart=aend=at;
}
else
{
aend->next=at;
at->previous=aend;
aend=at;
}
m++;
}
}
TagNode *t;
t=new TagNode;
t->content=str;
t->isTag=true;
t->parent=current;
t->attributeStart=astart;
t->attributeEnd=aend;
t->attributesCount=numberOfSplits-1;
if(start==NULL)
{
start=end=t;
}
else
{
if(current->start==NULL)
{
current->start=current->end=t;
}
else
{
current->end->next=t;
t->previous=end;
current->end=t;
}
current->childCount++;
}
current=t;
}
continue;
}
```

```
sb.append(ch);
}
}
void traverseDataStructure()
{
int i;
cout<<"******Traversing the data structure*******"<<endl;
int numberOfTabs=0;
char *str;
int x;
TagNode *t,*j;
AttributeNode *at;
j=NULL;
t=start;
while(1)
{
if(t==NULL)
{
if(j==NULL) break;
str=getEndTag(j->content);
numberOfTabs--;
for(i=1;i<=numberOfTabs;i++)
{
cout<<"\t";
}
cout<<str<<endl;
delete [] str;
t=j;
j=j->parent;
t=t->next;
if(t==NULL) continue;
}
for(i=1;i<=numberOfTabs;i++)
{
cout<<"\t";
}
cout<<t->content<<endl;
if(t->isTag)
{
numberOfTabs++;
at=t->attributeStart;
if(at!=NULL)
{
cout<<"Number of attribute : "<<t->attributesCount<<endl;
}
while(at!=NULL)
{
```

```
cout<<"("<<at->name<<","<<at->value<<")  ";
at=at->next;
if(at==NULL)
{
cout<<endl;
}
}
j=t;
t=t->start;
}
else
{
t=t->next;
}
}
}
void clearDataStructure()
{
if(start!=NULL)
{
delete start;
start=end=current=NULL;
}
}
int main()
{
int fileParsed=parseFile((char *)"test.txt");
if(fileParsed)
{
cout<<"File parsed"<<endl;
traverseDataStructure();
clearDataStructure();
}
else
{
cout<<"Cannot parse file"<<endl;
}
return 0;
}
```

Assignment create
TagNode * parseTagString(char *str,int &isStartAndEndTag)