COURSE PROJECT

XML PARSER

ABSTRACT

XML parser is a program that provides interface for client applications to work with XML documents. It checks for proper format of the XML document. The parser aims to check the syntax for a well- defined file and it is used all round in the software. The parser reads the document and analyses the structure of the document. It detects errors during translation and gives suggestions on how to correct them.

Introduction

XML

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML PARSER

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. The goal of a parser is to transform XML into a readable code.

Following are the various types of parsers which are commonly used to parse XML documents.

- Dom Parser – Parses an XML document by loading the complete contents of the document and creating its complete hierarchical tree in memory.

- SAX Parser – Parses an XML document on event-based triggers. Does not load the complete document into the memory.

- JDOM Parser – Parses an XML document in a similar fashion to DOM parser but in an easier way.

- StAX Parser – Parses an XML document in a similar fashion to SAX parser but in a more efficient way.

- Xpath Parser – Parses an XML document based on expression and is used extensively in conjunction with XSLT.

- DOM4J Parser – A java library to parse XML, Xpath, and XSLT using Java Collections Framework. It provides support for DOM, SAX, and JAXP.

Advantages of XML

- XML uses human, not computer, language. XML is readable and understandable, even by novices, and no more difficult to code than HTML.

- XML is completely compatible with Java and 100% portable. Any application that can process XML can use your information, regardless of platform.

- XML is extendable. Create your own tags, or use tags created by others, that use the natural language of your domain, that have the attributes you need, and that makes sense to you and your users.

Module Description

In the XML parser program, there are different functions which is implemented using stack which checks for errors in the program. The program checks whether the syntax for opening and closing tag is same, whether it contains any character that are not allowed in the syntax, etc. It will read an XML file and check for errors in the code if any error found then it displays the error with line number else shows the compilation is successful.

The main functions used are stringanalyser, stringanalysertag, and charanalyser. The functions such as push, pop, pusher, and poper are used for stack operations. In Stringanalyser, function is to analyse each and every tokens. First of all, a while loop is introduced to move till end of file. It checks each and every token. If there is no content in between the opening (<) and closing (>) tags it prints invalid tags, if it is a blank space it shows error and it also checks for space Infront of tags. If root is detected, it will invoke the pusher function and reads the string between the tags and also check for the closing tags. In stringanalysertag, a while loop is used to move the cursor from the beginning to the end of the file. If the first and second character is an opening tag then it shows error and if there is two closing tag then also it shows error. It also shows error if there is any missing of opening and closing tag. The function charanalyser is used to check whether the string entered inside the tag to check if the Tag name entered is valid or not. If, the Tag name is valid the functions push 'Valid_Tag_Name' and pop 'Valid_Tag_name' are invoked else, error message will be displayed. Operations are performed using stack. Stack is a linear data structure which follows a particular order in which the operations are performed. It works in LIFO(Last In First Out) maner. Which includes pusher, poper, pop, and push. In pop operation, it initially checks whether top == -1. If yes then it prints stack underflow else decrement the value of top. In push function it has two arguments a character value, item and an integer value, no. It initially checks whether the top value >= SIZE, if yes then print stack overflow else increment the value of top and assign stack [top].s as item and no stack[top].line as no, where s and line are members of structure ele. The pusher function is used add an item in the stack. If the stack is full, then it is said to be an Overflow condition. Here, we push or insert 'Valid_Tag_name' into the stack whenever this name occurs. The poper function is used to removes items from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition. Here, we pop or delete 'Valid_Tag_name' out of the stack whenever this name occurs.

Source Code

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#define SIZE 100000

struct Nest

{

   int line;

   char tag1[20];

   char tag2[20];

};

struct Nest nest[50];

struct ele1

{

   int ln;

   char str[30];

};

struct ele1 stack1[500];

int top1=-1;

struct string1

{

   int line;

   char no[20];
};
struct string2
{
   int line;
   char yes[20];
};
int flag=0,fl=0,rootDetect=0,rootLineNo,lastLineNo,fla=0;
int m=0;
char rootOpen[20];
int k=0,k1=0,okRule;
struct string1 str1[500];
struct string2 str2[500];
int top=-1;
//char stack[SIZE];
struct ele
{
   int line;
   char s;
};
struct ele stack[SIZE];
void pop();
void push(char item,int no);
void StringAnalyser();
void StringAnalyserTag();
void poper(char str[],int no);
void pusher(char st[],int no);
void CharAnalyser(char element[],int LineNo);

void RootAnalyser();

struct XMLDOC

{

   char source[SIZE];

}doc;

int main()

{

   //opening the xml file

   FILE *fp;

   fp = fopen(“TheFile.xml”,”r”);

   if (fp==NULL)

   {

      printf(“CouldNot load file”);

   }

   //setting the read cursor to the end of the file

   fseek(fp,0,SEEK_END);

   //to find the size of the file

   int size = ftell(fp);

   //set the read cursor to the start of the file

   fseek(fp,0,SEEK_SET);

   int count = fread(doc.source,sizeof(char),size,fp);//read the file content to the char pointer doc.source

   int len = strlen(doc.source);

   doc.source[len+1]=’\0’;

   fclose(fp);
   printf("\nThe no of Elemnts read = ");

   printf("%d\n",count);

   StringAnalyserTag();

   if (flag==0)

   {

      StringAnalyser();

   }

     if (top1>=-1)

     {

        for (int i = 0; i <=top1; i++)

        {

           fla=1;

           printf("\nLine No : %d -> Tag <%s> do not have matching </CLOSE>
tag",stack1[i].ln,stack1[i].str);

        }

     }

        if (rootLineNo!=lastLineNo)

        {

           printf("\nError : Root element not Found");

        }

        else if(flag==0 &&fla!=1)

        {

           printf("\nSuccessfully Compiled..NO ERRORS");

        }

}

void poper(char str3[],int no)
```

```
{
    if (top1==-1)
    {
      fla=1;
      printf("\nLine No : %d -> Tag </%s> do not have matching <OPEN> tag
\n",no,str3);
      fl=1;
    }
    else
    {
      int i=0,f=0;
      int value = strcmp(str3,stack1[top1].str);
      if (value==0)
      {
        lastLineNo = stack1[top1].ln;
        top1--;
      }else
      {
        for (i = 0; i <= top1; i++)
        {
          int val = strcmp(str3,stack1[i].str);
          if (val==0)
          {
            f=1;
            break;
          }
```

```
            }
            int val1 = strcmp(str3,stack1[top1].str);
            if (f==1&& val1!=0)
            {
                  printf("\nLine No : %d -> The closing Tag of <%s> should be in place of <%s>",no,stack1[i].str,stack1[top1].str);
                  top1--;
            }
            if (f==0)
            {
               fla=1;
               printf("\nLine No  : %d -> Tag </%s> do not have matching <OPEN> tag",no,str3);
               fl=1;
            }
            else
            {
               lastLineNo = stack1[i].ln;

                  for (i;i <top1;i++)
                  {
                     stack1[i] = stack1[i+1];
                  }
                  top1--;
            }
         }
      }
```

```
}
void pusher(char st[],intnum)
{
   if (top1>=SIZE-1)
   {
      printf("Stack overflow");
   }
   else
   {
      if (top1==-1)
      {
         top1++;
         strcpy(stack1[top1].str,st);
         stack1[top1].ln =num;
      }
      else
      {
         top1++;
         strcpy(stack1[top1].str,st);
         stack1[top1].ln =num;
      }

   }
}
void StringAnalyser()
{
```

```
int i=0,j=0,k1=0,lnNo=1,y1=0,y=0;

while (doc.source[i]!=’\0’)

{

   if (doc.source[i]==’\n’)

   {

      lnNo++;

   }

   if (doc.source[i]==’<’)

   {

      int check=0;

      if (doc.source[i+1]!=’/’)

      {

         if (doc.source[i+1]==’>’)

         {

            printf(“\nLine No : %d -> invalid tag <>”,lnNo);

            flag=1;

         }

         else

         {

            if (doc.source[i+1]==’ ‘)

            {

               i++;

               check=1;

               printf(“cool”);

            }

            else
```

```
{
   check=0;
}
i++;
while (doc.source[i]!=’>’ && doc.source[i]!=’ ‘)
{
   str1[k1].line=lnNo;
   str1[k1].no[y1] = doc.source[i];
   i++;
   y1++;
}
if (check==1)
{
   printf(“Line No : %d -> space not allowed at XVIeginning of Tag
%s”,lnNo,str1[k1].no);
   k1++;
   y1=0;
   fla=1;
   i++;
   continue;
}
CharAnalyser(str1[k1].no,lnNo);
if (okRule!=3)
{
   if (rootDetect==0)
   {
```

```
                        strcpy(rootOpen,str1[k1].no);
                        rootLineNo = lnNo;
                        rootDetect=1;
                    }
                    pusher(str1[k1].no,lnNo);
                }
                k1++;
                y1=0;
            }
        }
        else
        {
            if (doc.source[i+2]==’>’)
            {
                printf(“\nLine No : %d -> invalid tag </>”,lnNo);
                flag=1;
            }
            else
            {
                i=i+2;
                int check1=0;
                while (doc.source[i]!=’>’)
                {
                    str2[k].line=lnNo;
                    str2[k].yes[y] = doc.source[i];
                    i++;
                    y++;
                    if (doc.source[i]==’\n’ || doc.source[i]==’<’ || doc.source[i]=
        =’‘)
                    {
                    check1 = 1;
                    fla=1;
                    printf(“\nLine No : %d -> Invalid </CLOSE> Tag %s”,lnNo,str2[k].y
        es);
                    break;
                    }else
                    {
                        check1=0;
                    }
                }
                CharAnalyser(str2[k].yes,lnNo);
                if(okRule!=3&&check1==0)
                {
                    poper(str2[k].yes,lnNo);
                }
                k++;
                y=0;
            }
        }
      }
      i++;
    }
        }
```

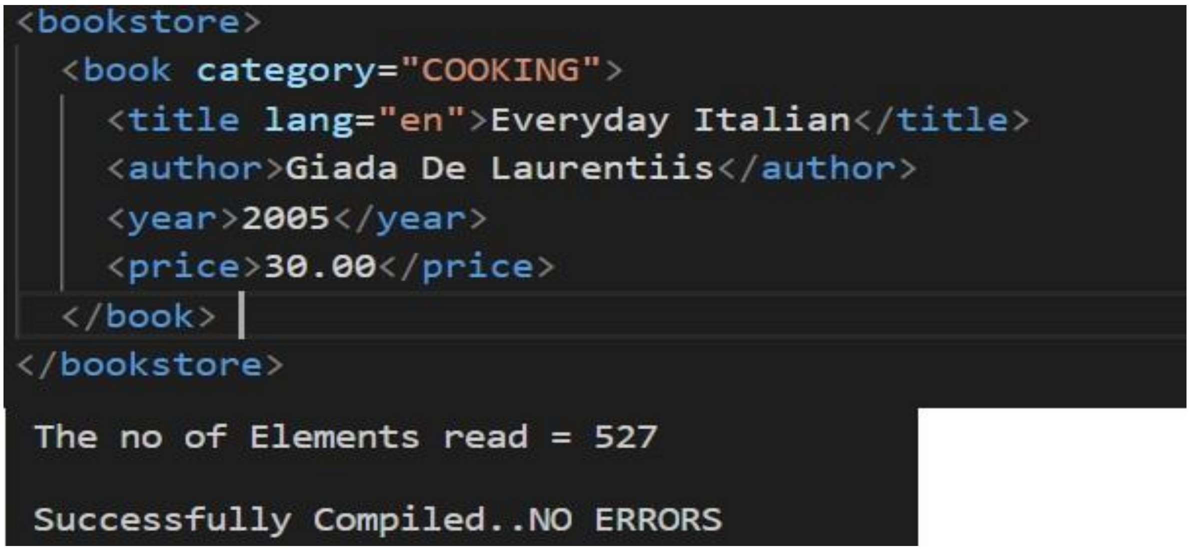
```
void StringAnalyserTag()
{
   int i=0,lineNo=1;
      while (doc.source[i]!='\0')
      {
         if (doc.source[i]=='\n')
         {
            lineNo++;
         }
        if (doc.source[i]=='<')
         {
            if (doc.source[i+1]=='<')
            {
               flag=1;
               printf("\nLine No : %d -> UnWanted Tags '<' detected",lineNo);
            }
            else
            {
               push(doc.source[i],lineNo);
            }
         }
         else if (doc.source[i]=='>')
         {
            if (doc.source[i+1]=='>')
            {
               flag=1;
```

```
                    printf("\nLine No : %d -> UnWanted Tags '>' detected",lineNo);
                }
                else
                {
                    if (stack[top].s == '<' && stack[top].line==lineNo)
                    {
                        pop();
                    }
                    else
                    {
                        flag = 1;
                        printf("\nLine No : %d -> Tag not closed '<' missing",lineNo);
                    }
                }
            }
            i++;
        }
        if (top!=-1)
        {
            flag=1;
            for (int i = 0; i <= top; i++)
            {
            printf("\nLine No : %d -> Tag not closed '>' missing",stack[i].line);
            }
        }
}
```

```
void pop()
{
   if (top==-1)
   {
      printf("stack underflow");
   }
   else
   {
      top--;
   }
}
void push(char item,int no)
{
   if (top>=SIZE-1)
   {
      printf("Stack overflow");
   }
   else
   {
      if (top==-1)
      {
         top++;
         stack[top].s = item;
         stack[top].line= no;
      }
      else
```

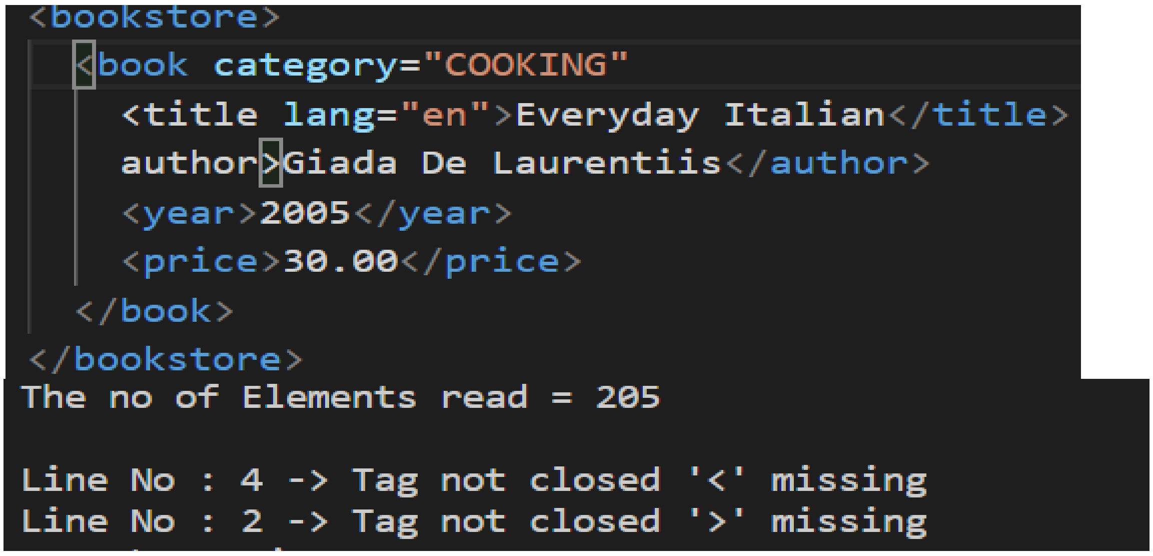
```
        {
           top++;
           stack[top].s=item;
           stack[top].line = no;
        }
    }
}
void CharAnalyser(char element[],intLineNo)
{
    if(element[0]=='_'||((element[0]>='a'&&element[0]<='z')||(eleme
nt[0] >= 'A' && element[0] <= 'Z')))
    {
        okRule = 2;
    }
    else
    {
        printf("\nLine No : %d -> Tag <%s> not allowed",LineNo,element);
        flag=1;
        okRule = 3;
    }
}
```

Snap Shot



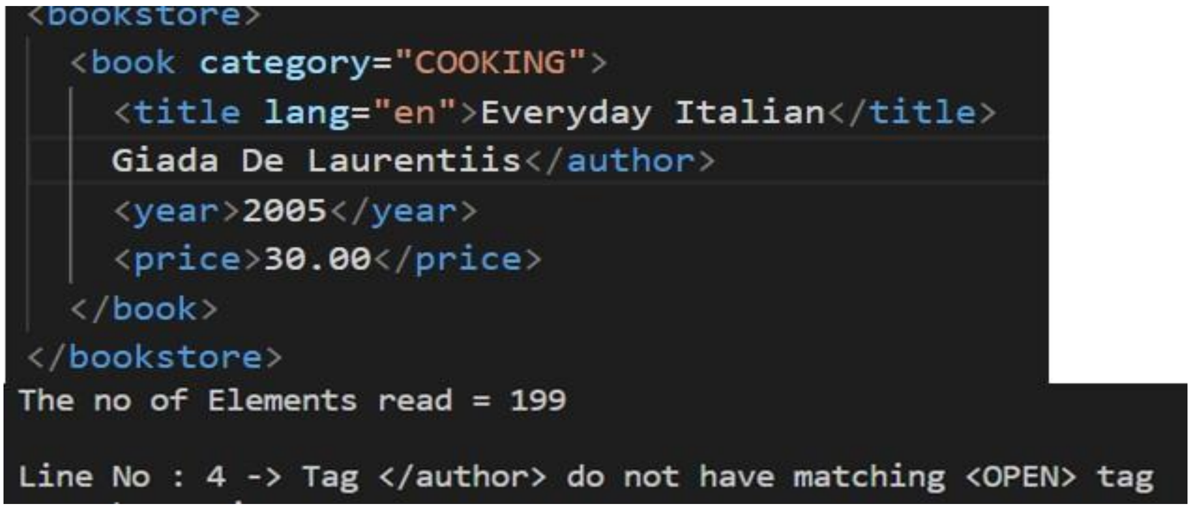
```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

```
 The no of Elements read = 527

 Successfully Compiled..NO ERRORS
```



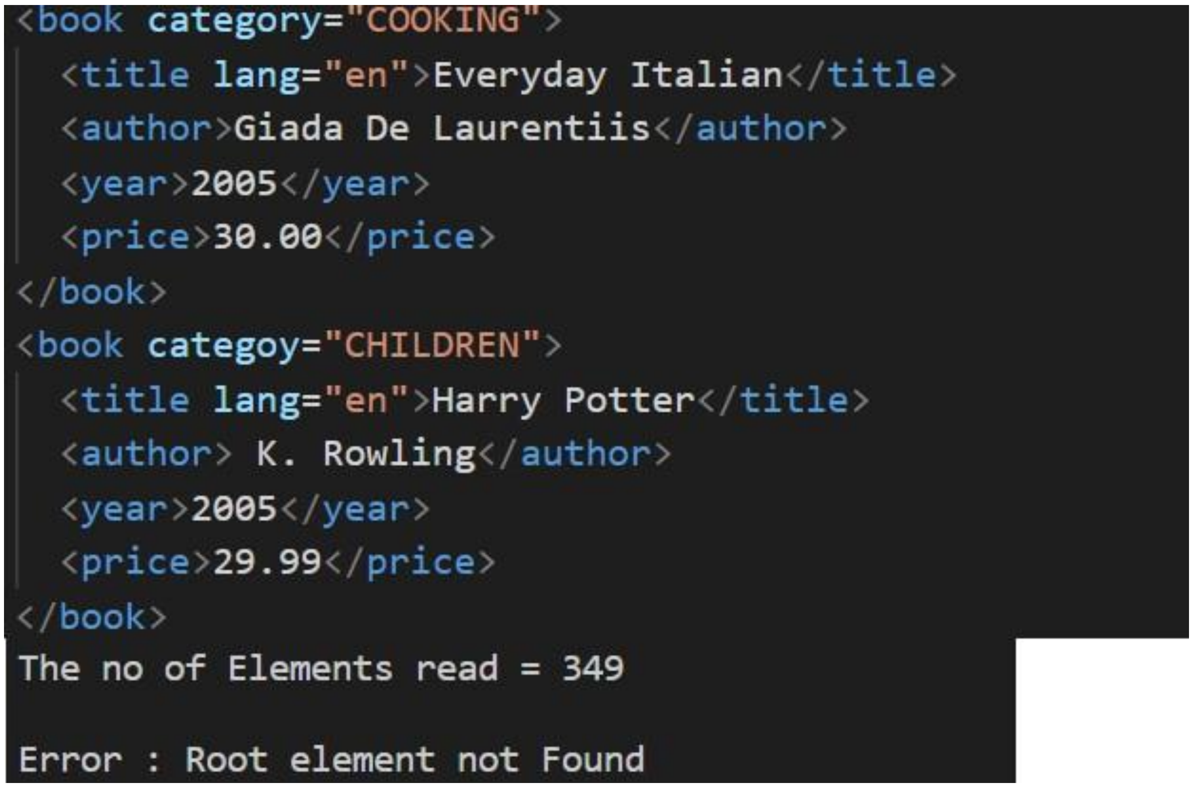
```
 <bookstore>
   <book category="COOKING"
     <title lang="en">Everyday Italian</title>
     author>Giada De Laurentiis</author>
     <year>2005</year>
     <price>30.00</price>
   </book>
 </bookstore>
```

```
The no of Elements read = 205

Line No : 4 -> Tag not closed '<' missing
Line No : 2 -> Tag not closed '>' missing
```

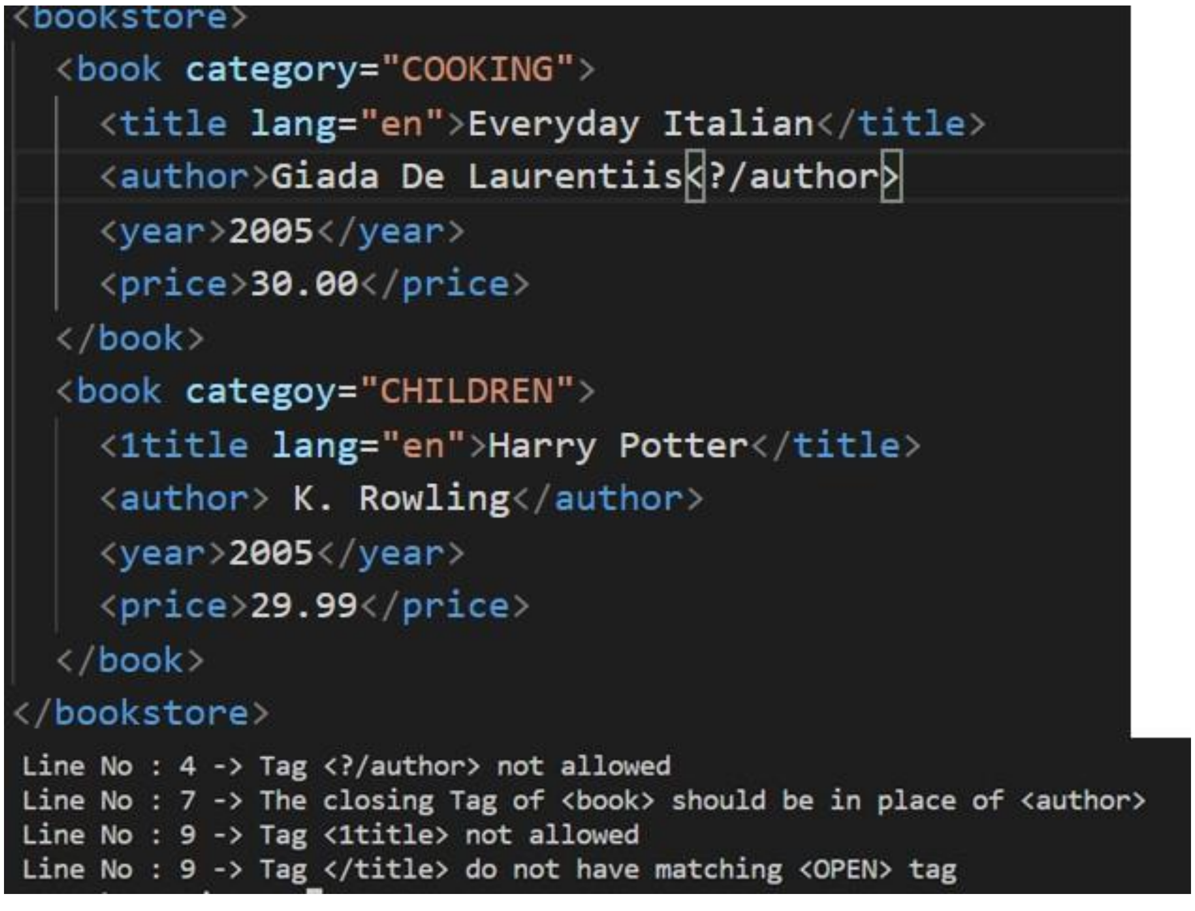


```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

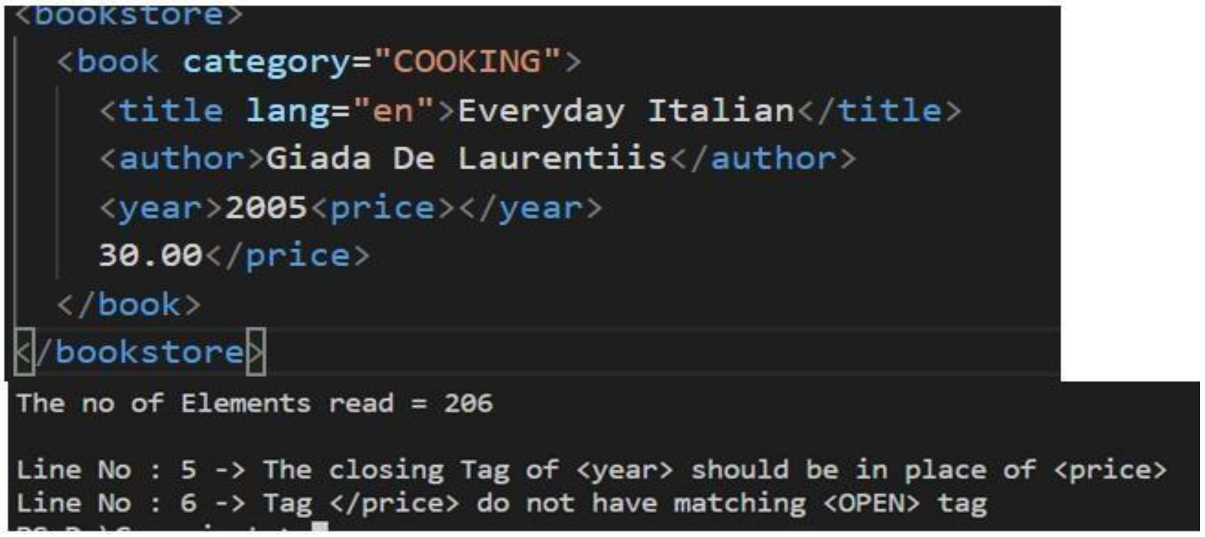
```
The no of Elements read = 199

Line No : 4 -> Tag </author> do not have matching <OPEN> tag
```



```
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book categoy="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author> K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
The no of Elements read = 349

Error : Root element not Found
```

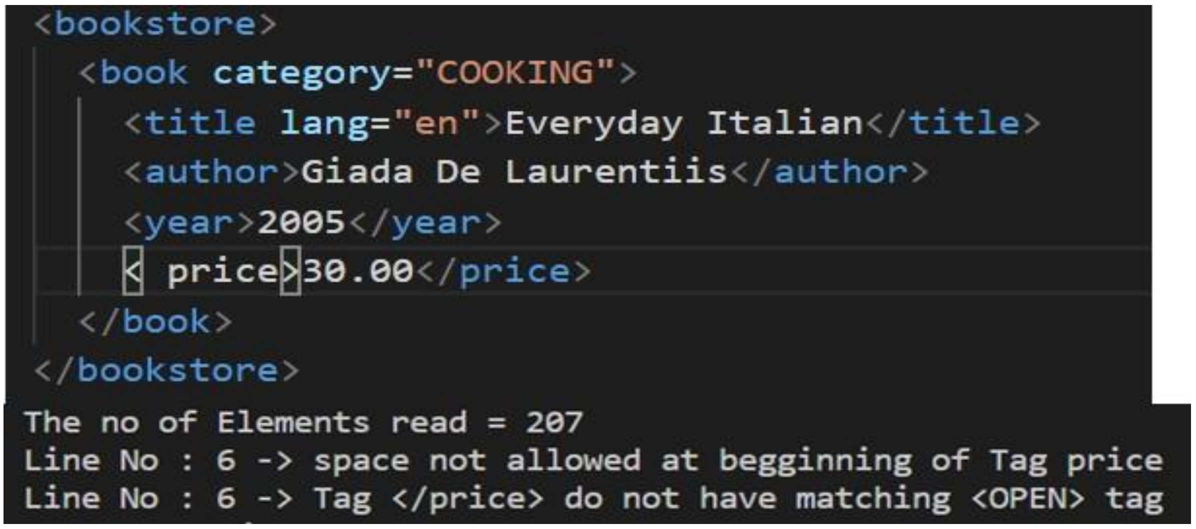


```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis<?/author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book categoy="CHILDREN">
    <1title lang="en">Harry Potter</title>
    <author> K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
Line No : 4 -> Tag <?/author> not allowed
Line No : 7 -> The closing Tag of <book> should be in place of <author>
Line No : 9 -> Tag <1title> not allowed
Line No : 9 -> Tag </title> do not have matching <OPEN> tag
```



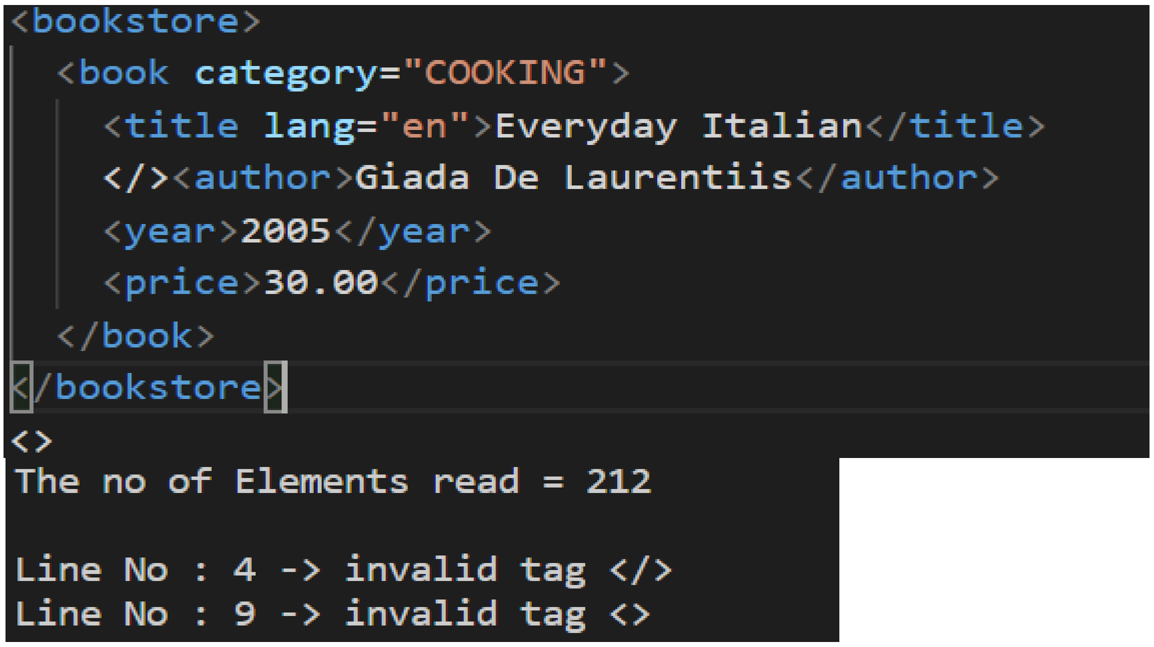
```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005<price></year>
    30.00</price>
  </book>
</bookstore>
```

```
The no of Elements read = 206

Line No : 5 -> The closing Tag of <year> should be in place of <price>
Line No : 6 -> Tag </price> do not have matching <OPEN> tag
```



```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    < price>30.00</price>
  </book>
</bookstore>
```

```
The no of Elements read = 207
Line No : 6 -> space not allowed at beggining of Tag price
Line No : 6 -> Tag </price> do not have matching <OPEN> tag
```



```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    </><author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
<>
```

```
The no of Elements read = 212

Line No : 4 -> invalid tag </>
Line No : 9 -> invalid tag <>
```

Features of the project

- It’s a simple parser but yet fully loaded with many features. It is implemented using basic data structures like stack and arrays.
- This parser displays most of the syntactical errors in the XML file.
- We also used file system in this parser which loads the xml file contents to a character array.
- This parser can detect errors like missing tags, unmatched tags, unwanted tags, improperly nested tags, special characters and white spaces in the tag names and many more.
- The main feature of our parser is that not only it displays the specific error but it also displays the correct line number at which the error has occurred.
- This parser also gives suggestions on how to fix the errors.
- It’s very fast parser that displays errors no matter how heavy the xml file is.
- Instead of line by line scanning we scanned the whole xml file from left to right so number of loops has reduced and hence improves performance