# XML Parser

| NAME | ROLL | REGISTRATION NUMBER |
|---|---|---|
| Abhinav Agrawal | 37 | 130911304 |

## DESCRIPTION

Extensible Markup Language ('XML') is a markup language that defines a set of rules for encoding documents in a File format which is both human-readable and machine-readable.

The design goals of XML emphasize simplicity, generality and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures.

XML data is known as self-describing or self-defining, meaning that the structure of the data is embedded with the data, thus when the data arrives there is no need to pre-build the structure to store the data; it is dynamically understood within the XML. The XML format can be used by any individual or group of individuals or companies that want to share information in a consistent way. XML is actually a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML), which is the standard to create a document structure.

The basic building block of an XML document is an element, defined by tags. An element has a beginning and an ending tag. All elements in an XML document are contained in an outermost element known as the root element. XML can also support nested elements, or elements within elements. This ability allows XML to support hierarchical structures. Element names describe the content of the element, and the structure describes the relationship between the elements.

For example, XML documents can be very simple, such as the following:

```
<?xml version="1.0" standalone="yes"?>
<conversation>
    <greeting>Hello, world!</greeting>
    <response>Stop the planet, I want to get off!</response>
</conversation>
```

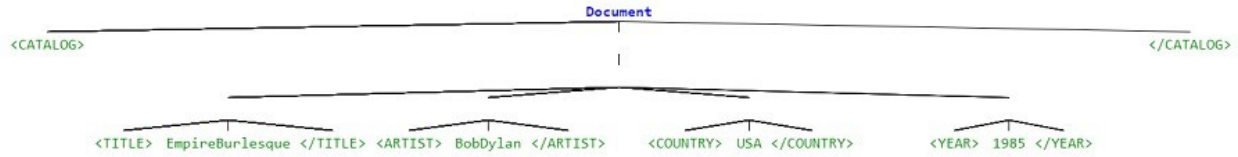
Applications for XML are endless. For example, computer makers might agree upon a standard or common way to describe the information about a computer product (processor speed, memory size, and so forth) and then describe the product information format with XML code. Such a standard way of describing data would enable a user to send an intelligent agent (a program) to each computer maker's Web site, gather data, and then make a valid comparison.

XML's power resides in its simplicity. It can take large chunks of information and consolidate them into an XML document - meaningful pieces that provide structure and organization to the information.

**TERMINALS:** VERSION, START, ATTDEF, ENDDEF, EQ, SLASH, CLOSE, END, COMMENT, NAME

**NON-TERMINALS**: document, prolog , version_opt, encoding_opt, misc_seq_opt, misc, attribute_decl, element, empty_or_content, content, name_opt, attribute_seq_opt, attribute

## Annotation Parse Tree:




## IMPLEMENTATION

### LEX

```
%{
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include "y.tab.h"
static int keep;

//Storing string in the buffer
```

```
static char* word(char *s)
{
  char *buf;
  int i, k;
  for (k = 0; isspace(s[k]) || s[k] == '<'; k++) ;
  for (i = k; s[i] && ! isspace(s[i]); i++) ;
  buf = (char*)malloc((i - k + 1) * sizeof(char));
  strncpy(buf, &s[k], i - k);
  buf[i - k] = '\0';
  return buf;
}
%}


nl              (\r\n|\r|\n)
ws              [ \t\r\n]+
open            {nl}?"<"
close           ">"{nl}?
namestart       [A-Za-z\200-\377_]
namechar        [A-Za-z\200-\377_0-9.-]
esc             "&#"[0-9]+";"|"&#x"[0-9a-fA-F]+";"
name            {namestart}{namechar}*
data            ([^<\n&]|\n[^<&]|\n{esc}|{esc})+
comment                 {open}"!--"([^-]|"-"[^-])*"--"{close}
string          \"([^"&]|{esc})*\"|\'([^'&]|{esc})*\'
version         {open}"?XML-VERSION 1.0?"{close}

encoding        {open}"?XML-ENCODING"{ws}{name}{ws}?"?"{close}
attdef          {open}"?XML-ATT"
```

```
%s CONTENT
%%
<INITIAL>{ws}                   {/* skip */}
<INITIAL>{version}{return VERSION;}
<INITIAL>{encoding}             {yylval.s = word(yytext + 14); return ENCODING;}
<INITIAL>"/"            {return SLASH;}
<INITIAL>"="                    {return EQ;}
<INITIAL>{close}        {BEGIN(CONTENT); return CLOSE;}
<INITIAL>{name}                 {yylval.s = strdup(yytext); return NAME;}
<INITIAL>{string}       {yylval.s = strdup(yytext); return VALUE;}
<INITIAL>"?"{close}             {BEGIN(keep); return ENDDEF;}
{attdef}                {keep = YY_START; BEGIN(INITIAL); return ATTDEF;}
{open}{ws}?{name}{BEGIN(INITIAL); yylval.s= word(yytext); return START;}
{open}{ws}?"/"                  {BEGIN(INITIAL); return END;}
{comment}               {yylval.s = strdup(yytext); return COMMENT;}
<CONTENT>{data}                 {yylval.s = strdup(yytext); return DATA;}
.                       {fprintf(stderr, "!error due to (%c)\n", *yytext);}
{nl}                    {/* skip, must be an extra one at EOF */;}
```

**YACC**

```
%{
#include <string.h>
#include <stdio.h>
#include <iostream>
using namespace std;
extern FILE *yyin;
void yyerror(char *msg);
```

```
int yylex();
%}
%union {char *s;}
%token VERSION ATTDEF ENDDEF EQ SLASH CLOSE END
%token <s> ENCODING NAME VALUE DATA COMMENT START
%type <s> name_opt

%%
document
 : prolog element misc_seq_opt
 ;
prolog
 : version_opt encoding_opt
   misc_seq_opt
 ;
version_opt
 : VERSION                    {printf("<?XML-VERSION 1.0?>\n");}

 ;
encoding_opt
 : ENCODING                          {printf("<?XML-ENCODING %s?>\n",$1); free($1);}
 | /*empty*/
 ;
misc_seq_opt
 : misc_seq_opt misc
 | /*empty*/
 ;
misc
 : COMMENT                    {printf("%s", $1);}
 | attribute_decl
```

```
 ;
attribute_decl
 : ATTDEF NAME                      {printf("\n<?XML-ATT %s", $2);}
   attribute_seq_opt ENDDEF{printf("?>\n");}
 ;
element
 : START                            {printf("\n<%s", $1); free($1);}
   attribute_seq_opt
   empty_or_content
 ;

empty_or_content
 : SLASH CLOSE                      {printf("/>\n");}
 | CLOSE                        {printf(">\n");}
   content END name_opt CLOSE       {printf("\n</%s>\n", $5);}
 ;
content
 : content DATA                     {printf("%s", $2); free($2);}
 | content misc
 | content element
 | /*empty*/
 ;
name_opt
 : NAME                             {$$ = $1;}
 | /*empty*/                    {$$ = strdup("");}
 ;
attribute_seq_opt
 : attribute_seq_opt attribute
 | /*empty*/
 ;
```

```
attribute
 : NAME                                {printf(" %s", $1); free($1);}
 | NAME EQ VALUE            {printf(" %s=%s", $1, $3); free($1); free($3);}
 ;
%%

int yywrap(void)
{
  return 1;
}

void yyerror(char *msg)
{
  fprintf(stderr, "%s\n", msg);
}

int main(int argc, char *argv[])
{
        char *s;
        int no_errors = 0, no_error2 = 0, no_errors3 = 0;
        if (argc > 1) {
                s = argv[1];
                yyin = fopen(s, "r");
                yyparse();
        }
        cout << "\n**************ERROR AFTER
PARSING**************************\n";
        //lexerr(s);
        cout << "\n*****************************************************************\n";
        //yaccers(s);
```

```
    return 0;
}
```