Project Report

British to American English Converter Using Lex

By Vy Bui, Tuan Nguyen, Jonathan Fox

CSC 504 Compiler Design

The Catholic University of America

12/09/2013

Introduction

Although America and Great Britain and all former English speaking colonies of Great Britain share a common language, the spelling of words has two versions, American and British spelling. The biggest differences are with the ending syllables (British, American): -our, -or; -re, -er; -ce, -se; -xion, -ction; -ise, -ize; -yse, -yze; -ogue, -og; and -ae, -oe. While there are other differences such as doubled consonants (in both versions), dropped "e" (ageing and aging), and different spelling and pronunciations all together (aeroplane vs airplane). While the need to change one set to another it relatively minor there are some cases where it is useful to have a way to parse through a document and make changes. For example, the English portion of Wikipedia attracts users from all other the World and there occasional variants in spelling. Changing to one set standard of spelling would keep up an appearance of an edited encyclopedia.

Design

The design of the program is simple enough. The primary idea was to take a plain text file of British English spelling, use a Lexical analyzer to go through every string in the file, identify special ending character groups, and replace into a new text file. In the lex file used, `string.l`, the first step was to develop the regular expressions to identify the key spellings.

```
our [a-zA-Z][a-zA-Z]+our
re [a-zA-Z]+re
ce [a-zA-Z]+ce
xion [a-zA-Z]+xion
ise [a-zA-Z]+ise[s|d]?
isation [a-zA-Z]+isation
yse [a-zA-Z]+yse[s|d]?
ogue [a-zA-Z]+ogue
ae [a-zA-Z]+ae[a-zA-Z]+
oe [a-zA-Z]+oe[a-zA-Z]+
eol \n
```

As shown above, those regular expressions identify the characters at the end of a string having a British spelling variant. Next was to come up with the functions of replacing substrings in the text, with the function aptly named `replace`.

```
char * replace(
    char const * const original,
    char const * const pattern,
    char const * const replacement
) {
  size_t const replen = strlen(replacement);
  size_t const patlen = strlen(pattern);
  size_t const orilen = strlen(original);

  size_t patcnt = 0;
  const char * oriptr;
  const char * patloc;
```

```
  // find how many times the pattern occurs in the original string
  for (oriptr = original; patloc = strstr(oriptr, pattern); oriptr =
patloc + patlen)
  {
    patcnt++;
  }

  {
    // allocate memory for the new string
    size_t const retlen = orilen + patcnt * (replen - patlen);
    char * const returned = (char *) malloc( sizeof(char) * (retlen +
1) );

    if (returned != NULL)
    {
      // copy the original string,
      // replacing all the instances of the pattern
      char * retptr = returned;
      for (oriptr = original; patloc = strstr(oriptr, pattern); oriptr
= patloc + patlen)
      {
        size_t const skplen = patloc - oriptr;
        // copy the section until the occurence of the pattern
        strncpy(retptr, oriptr, skplen);
        retptr += skplen;
        // copy the replacement
        strncpy(retptr, replacement, replen);
        retptr += replen;
      }
      // copy the rest of the string.
      strcpy(retptr, oriptr);
    }
    return returned;
  }
}
```

The function `replace` first defines three new variables: `original`, `pattern`, and `replacement` (as both character data and string length). The pattern is then analyzed to see how many times it occurs. Memory is allocated for the new string, the original string is copied and all instances of the pattern is replaced. After the `replace` function, there is the substring replacement implementation as shown below in the case of "-our."

```
{our} {
      char * const newstr = replace(yytext, "our", "or");
      if (newstr)
      {
        fprintf(fr,newstr);
              free(newstr);
      }
     }
```

Finally the code main function is:

```
int main(int argc, char *argv[])
{
        strcpy(fname,argv[1]);
        ff=fopen(fname,"r+");
        fr=fopen("result.txt","w+");
        yyin=ff;
        yylex();
        return(0);
}
```

Results

The text document used to contain the original text is `find.txt`.

```
//This project will translate British English to American English
Eg:
this colour is blue.
there is no black colour here.
the labour is in centre of the party.
our organisation realise that your advice is so true.
encyclopaedia are types of references.
He was paralysed since last year.
Complexion
analogue
```

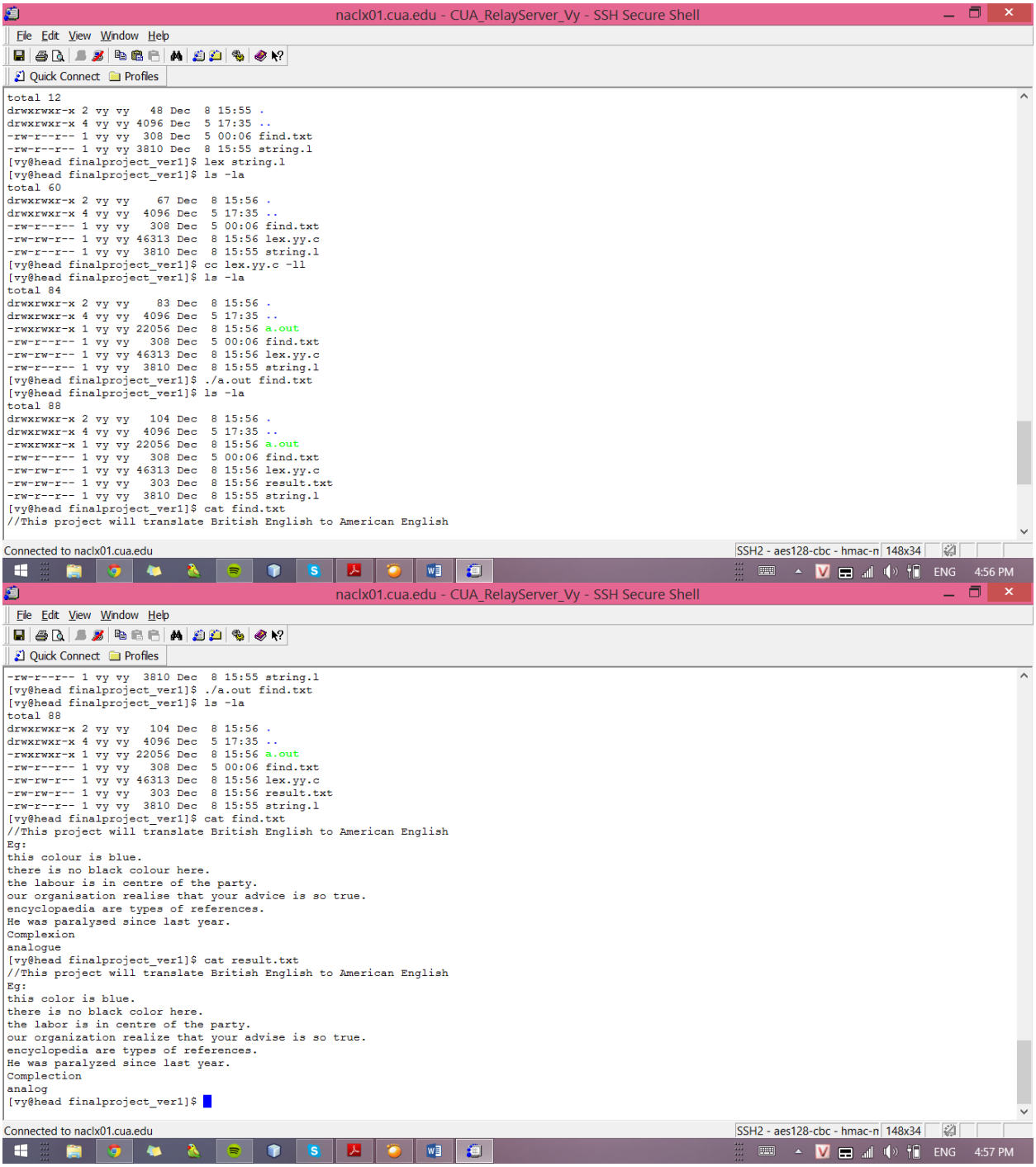
After running through the lexical analyzer, the output text file, result.txt is as follows:

```
//This project will translate British English to American English
Eg:
this color is blue.
there is no black color here.
the labor is in center of the party.
our organization realize that your advise is so true.
encyclopedia are types of references.
He was paralyzed since last year.
Complection
analog
```

The lexical analyzer is successful at parsing through the text file and identifying the key substrings containing the British English spelling. The only problem is the analyzer does not take an account the unstressed or stress nature of the ending vowel. In the case of –our, there are unstressed forms in colour, flavour, and harbour, but in other words ending in –our, there are stressed vowel which has unreduced pronunciation, such as contour, velour, and troubadour. A simple look up table as a case situation could settle this however, the size of the analyzer would increasing when having a large table of words to remember not to change. A similar case is was

not taken account in this iteration, is the –cre syllable. Words like acre, massacre, and mediocre, could get changed if a special rule is not written.

Demonstration



Conclusion

The project achieved its objectives, the analyzer is more than able to goes through a text document and identify the words with a specific final syllable. The only issues is the lack of

being able to know the difference between stressed and unstressed vowels. The exceptions, would need to be included somehow and is not an impossible task.

Appendix

String.l

```
%{
#include<stdio.h>
#include<string.h>
#include <stdlib.h>
FILE *ff,*fr;
char p[20],q[20],r[20],fname[20];

// function to replace substring
char * replace(
    char const * const original,
    char const * const pattern,
    char const * const replacement
) {
  size_t const replen = strlen(replacement);
  size_t const patlen = strlen(pattern);
  size_t const orilen = strlen(original);

  size_t patcnt = 0;
  const char * oriptr;
  const char * patloc;

  // find how many times the pattern occurs in the original string
  for (oriptr = original; patloc = strstr(oriptr, pattern); oriptr =
patloc + patlen)
  {
    patcnt++;
  }

  {
    // allocate memory for the new string
    size_t const retlen = orilen + patcnt * (replen - patlen);
    char * const returned = (char *) malloc( sizeof(char) * (retlen +
1) );

    if (returned != NULL)
    {
      // copy the original string,
      // replacing all the instances of the pattern
      char * retptr = returned;
      for (oriptr = original; patloc = strstr(oriptr, pattern); oriptr
= patloc + patlen)
      {
        size_t const skplen = patloc - oriptr;
        // copy the section until the occurence of the pattern
```

```
        strncpy(retptr, oriptr, skplen);
        retptr += skplen;
        // copy the replacement
        strncpy(retptr, replacement, replen);
        retptr += replen;
      }
      // copy the rest of the string.
      strcpy(retptr, oriptr);
    }
    return returned;
  }
}

%}

our [a-zA-Z][a-zA-Z]+our
re [a-zA-Z]+re
ce [a-zA-Z]+ce
xion [a-zA-Z]+xion
ise [a-zA-Z]+ise[s|d]?
isation [a-zA-Z]+isation
yse [a-zA-Z]+yse[s|d]?
ogue [a-zA-Z]+ogue
ae [a-zA-Z]+ae[a-zA-Z]+
oe [a-zA-Z]+oe[a-zA-Z]+
eol \n

%%
{our} {
      char * const newstr = replace(yytext, "our", "or");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }


        }
{re} {
      if
((strcmp("here",yytext)!=0)&(strcmp("there",yytext)!=0)&(strcmp("are",
yytext)!=0))
      {
      char * const newstr = replace(yytext, "re", "er");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }
           }
      else fprintf(fr,yytext);
```

```
        }

{ce} {
      if
((strcmp("advice",yytext)==0)|(strcmp("pratice",yytext)==0)|(strcmp("d
evice",yytext)==0)|(strcmp("licence",yytext)==0)|(strcmp("defence",yyt
ext)==0))
      {
      char * const newstr = replace(yytext, "ce", "se");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }
      else fprintf(fr,yytext);
     }
{xion} {
      char * const newstr = replace(yytext, "xion", "ction");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }
{ise} {
      char * const newstr = replace(yytext, "ise", "ize");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }
{yse} {
      char * const newstr = replace(yytext, "yse", "yze");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }

{isation} {
      char * const newstr = replace(yytext, "isation", "ization");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
```

```
      }

      }
{ogue} {
      char * const newstr = replace(yytext, "ogue", "og");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }
{ae} {
      char * const newstr = replace(yytext, "ae", "e");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }
{oe} {
      char * const newstr = replace(yytext, "oe", "e");
      if (newstr)
      {
        fprintf(fr,newstr);
             free(newstr);
      }

      }
{eol} {fprintf(fr,yytext);}
. {fprintf(fr,yytext);}
%%


int main(int argc,char *argv[])
{
        strcpy(fname,argv[1]);
        ff=fopen(fname,"r+");
        fr=fopen("result.txt","w+");
        yyin=ff;
        yylex();
        return(0);
}


yywrap()
{
  return(1);
}
```