PORTER STEMMING

AIM :

To Implement Porter-Stemming.

PROGRAM :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0

static char * b;
static int k,k0,j;
int cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return FALSE;
      case 'y': return (i==k0) ? TRUE : !cons(i-1);
      default: return TRUE;
   }
}
int m()
{  int n = 0;
   int i = k0;
   while(TRUE)
   {  if (i > j) return n;
      if (! cons(i)) break; i++;
   }
   i++;
   while(TRUE)
   {  while(TRUE)
      {  if (i > j) return n;
            if (cons(i)) break;
            i++;
      }
      i++;
      n++;
      while(TRUE)
      {  if (i > j) return n;
         if (! cons(i)) break;
         i++;
      }
      i++;
    }
}
int vowelinstem()
{  int i; for (i = k0; i <= j; i++) if (! cons(i)) return TRUE;
   return FALSE;
}
```

```
int doublec(int j)
{  if (j < k0+1) return FALSE;
   if (b[j] != b[j-1]) return FALSE;
   return cons(j);
}

int cvc(int i)
{  if (i < k0+2 || !cons(i) || cons(i-1) || !cons(i-2)) return FALSE;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return FALSE;
   }
   return TRUE;
}

int ends(char * s)
{  int length = s[0];
   if (s[length] != b[k]) return FALSE;
   if (length > k-k0+1) return FALSE;
   if (memcmp(b+k-length+1,s+1,length) != 0) return FALSE;
   j = k-length;
   return TRUE;
}
void setto(char * s)
{  int length = s[0];
   memmove(b+j+1,s+1,length);
   k = j+length;
}
void r(char * s) { if (m() > 0) setto(s); }

void step1ab()
{  if (b[k] == 's')
   {  if (ends("\04" "sses")) k -= 2; else
      if (ends("\03" "ies")) setto("\01" "i"); else
      if (b[k-1] != 's') k--;
   }
   if (ends("\03" "eed")) { if (m() > 0) k--; } else
   if ((ends("\02" "ed") || ends("\03" "ing")) && vowelinstem())
   {  k = j;
      if (ends("\02" "at")) setto("\03" "ate"); else
      if (ends("\02" "bl")) setto("\03" "ble"); else
      if (ends("\02" "iz")) setto("\03" "ize"); else
      if (doublec(k))
      {  k--;
         {  int ch = b[k];
            if (ch == 'l' || ch == 's' || ch == 'z') k++;
         }
      }
      else if (m() == 1 && cvc(k)) setto("\01" "e");
   }
}
void step1c() { if (ends("\01" "y") && vowelinstem()) b[k] = 'i'; }
void step2() { switch (b[k-1])
```

```
{
   case 'a': if (ends("\07" "ational")) { r("\03" "ate"); break; }
             if (ends("\06" "tional")) { r("\04" "tion"); break; }
             break;
   case 'c': if (ends("\04" "enci")) { r("\04" "ence"); break; }
             if (ends("\04" "anci")) { r("\04" "ance"); break; }
             break;
   case 'e': if (ends("\04" "izer")) { r("\03" "ize"); break; }
             break;
   case 'l': if (ends("\03" "bli")) { r("\03" "ble"); break; }
             if (ends("\04" "alli")) { r("\02" "al"); break; }
             if (ends("\05" "entli")) { r("\03" "ent"); break; }
             if (ends("\03" "eli")) { r("\01" "e"); break; }
             if (ends("\05" "ousli")) { r("\03" "ous"); break; }
             break;
   case 'o': if (ends("\07" "ization")) { r("\03" "ize"); break; }
             if (ends("\05" "ation")) { r("\03" "ate"); break; }
             if (ends("\04" "ator")) { r("\03" "ate"); break; }
             break;
   case 's': if (ends("\05" "alism")) { r("\02" "al"); break; }
             if (ends("\07" "iveness")) { r("\03" "ive"); break; }
             if (ends("\07" "fulness")) { r("\03" "ful"); break; }
             if (ends("\07" "ousness")) { r("\03" "ous"); break; }
             break;
   case 't': if (ends("\05" "aliti")) { r("\02" "al"); break; }
             if (ends("\05" "iviti")) { r("\03" "ive"); break; }
             if (ends("\06" "biliti")) { r("\03" "ble"); break; }
             break;
   case 'g': if (ends("\04" "logi")) { r("\03" "log"); break; }

} }

void step3() { switch (b[k])
{
   case 'e': if (ends("\05" "icate")) { r("\02" "ic"); break; }
             if (ends("\05" "ative")) { r("\00" ""); break; }
             if (ends("\05" "alize")) { r("\02" "al"); break; }
             break;
   case 'i': if (ends("\05" "iciti")) { r("\02" "ic"); break; }
             break;
   case 'l': if (ends("\04" "ical")) { r("\02" "ic"); break; }
             if (ends("\03" "ful")) { r("\00" ""); break; }
             break;
   case 's': if (ends("\04" "ness")) { r("\00" ""); break; }
             break;
} }
void step4()
{
switch (b[k-1])
   {  case 'a': if (ends("\02" "al")) break; return;
      case 'c': if (ends("\04" "ance")) break;
                if (ends("\04" "ence")) break; return;
```

```
      case 'e': if (ends("\02" "er")) break; return;
      case 'i': if (ends("\02" "ic")) break; return;
      case 'l': if (ends("\04" "able")) break;
                if (ends("\04" "ible")) break; return;
      case 'n': if (ends("\03" "ant")) break;
                if (ends("\05" "ement")) break;
                if (ends("\04" "ment")) break;
                if (ends("\03" "ent")) break; return;
      case 'o': if (ends("\03" "ion") && (b[j] == 's' || b[j] == 't')) break;
                if (ends("\02" "ou")) break; return;
                case 's': if (ends("\03" "ism")) break; return;
      case 't': if (ends("\03" "ate")) break;
                if (ends("\03" "iti")) break; return;
      case 'u': if (ends("\03" "ous")) break; return;
      case 'v': if (ends("\03" "ive")) break; return;
      case 'z': if (ends("\03" "ize")) break; return;
      default: return;
    }
    if (m() > 1) k = j;
}

void step5()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}
int stem(char * p, int i, int j)
{  b = p; k = j; k0 = i;
   if (k <= k0+1) return k;
   step1ab(); step1c(); step2(); step3(); step4(); step5();
   return k;
}
static char * s;
#define INC 50
static int i_max = INC;
void increase_s()
{  i_max += INC;
   {  char * new_s = (char *) malloc(i_max+1);
      { int i; for (i = 0; i < i_max; i++) new_s[i] = s[i]; }
      free(s); s = new_s;
   }
}

#define UC(ch) (ch <= 'Z' && ch >= 'A')
#define LC(ch) (ch <= 'z' && ch >= 'a')
#define LETTER(ch) (UC(ch) || LC(ch))
#define FORCELC(ch) (ch-('A'-'a'))

void stemfile(FILE * f)
```

```
{
while(TRUE)
  {
    int ch = getc(f);
    if (ch == EOF) return;
    if (LETTER(ch))
    {  int i = 0;
       while(TRUE)
       {  if (i == i_max) increase_s();

          if UC(ch) ch = FORCELC(ch);
          s[i] = ch; i++;
          ch = getc(f);
          if (!LETTER(ch)) { ungetc(ch,f); break; }
       }
       s[stem(s,0,i-1)+1] = 0;
       printf("%s",s);
    }
    else putchar(ch);
  }
}

int main(int argc, char * argv[])
{  int i;
  s = (char *) malloc(i_max+1);
  for (i = 1; i < argc; i++)
  {  FILE * f = fopen(argv[i],"r");
    if (f == 0) { fprintf(stderr,"File %s not found\n",argv[i]); exit(1); }
    stemfile(f);
  }
  free(s);
  return 0;
}
```

RESULT :

The Program “Porter-Stemming” Implemented Successfully And Required Output Is Obtained.

OUTPUT :

Input : - A text file named message.txt

message.txt : - Numerical Methods and Programing

output : - numer method and program