```
Script started on 2023-07-13 22:11:28-05:00 [TERM="xterm" TTY="/dev/pts/1" COLUMNS=
bj94684@ares:~$ pwd
/home/students/bj94684
bj94684@ares:~$ cat cw.info
/*****************************************************************************
 *                                                                           *
 *  NAME:  Jose Barron                          CLASS:  CSC122-002           *
 *                                                                           *
 *          Lab: Where, oh where,                                            *
 *               has my little SHEEP gone?!       Level:   6                *
 *       Option:                                  Level: + 0                *
 *                                          Total Level:   6.0               *
 *                                                                           *
 * This program is designed to generate a crossword from a list of words.    *
 * The list of words are from a user file. The crossword is first            *
 * genrated empty, and the size of the grid is dependent on the largest      *
 * word found in the list. Then through the use of the custom crossword.h    *
 * library the crossword is filled in with the words given by the user.      *
 * The crossword is generated by placing the first word in the crossword     *
 * in some random position and orientation, then the words are added         *
 * one by one in random positions and orientations, and if all the words     *
 * are in the crossword then it displays the crossword, but if not then      *
 * the code loops until all the words are in the crossword.                  *
*****************************************************************************/
bj94684@ares:~$ show-code crossword.h
```

crossword.h:

```
  1  #ifndef CROSSWORD_H_INC
  2  #define CROSSWORD_H_INC
  3
  4  #include<iostream>
  5  #include<string>
  6  #include<vector>
  7  #include<ctime>
  8  #include<cctype>
  9  #include <cstdlib>
 10
 11  inline short rand_num(short min, short max)
 12  {
 13      return static_cast<short>( rand()%(max-min+1) + min );
 14  }
 15
 16
 17  inline void display(std::vector<std::vector<char>> cw)
 18  {
 19      for (std::vector<std::vector<char> >::size_type row = 0;
 20             row != cw.size(); ++row)
 21        {
 22            for (std::vector<char>::size_type col = 0;
 23                 col != cw[row].size(); ++col)
 24          {
 25              std::cout << cw[row][col] << ' ';
 26          }
 27          std::cout << '\n';
 28        }
 29  }
 30
 31  inline std::vector<std::vector<char>> generate_cw(std::vector<std::vector<c
 32  {
 33      std::vector<std::vector<char>>cw1 = cw;
 34      cw1.resize(r);
 35      for (std::vector<std::vector<char> >::size_type row = 0;
 36           row != cw1.size(); ++row)
 37      {
 38          cw1[row].resize(col);
 39      }
 40      return cw1;
 41  }
 42
 43  inline std::string reverse_word(std::string s)
 44  {
 45      std::string reversedWord;
 46      std::string::size_type i = s.length() - 1;
 47      while( static_cast<short>(i) >= 0 )
 48      {
 49          reversedWord += s[i];
 50          i = i - 1;
 51      }
 52      return reversedWord;
 53  }
 54  inline std::vector<std::vector<char>> fill_cw(std::vector<std::vector<char>
 55  {
 56      std::vector<std::vector<char>>cw1 = cw;
 57      for (auto & r : cw1)
 58      {
 59          for (auto & c : r)
 60          {
 61              c = t;
 62          }
 63      }
 64      return cw1;
 65  }
 66
 67  std::vector<std::vector<char>> fill_cw_w_l(std::vector<std::vector<char>> c
 68                                             char t);
 69  std::vector<std::vector<char>> gen_cw_ans(std::vector<std::vector<char>> cw
 70                                             char t);
 71
 72
 73  bool vertical (std::vector<std::vector<char>> cw, short row, short column,
 74                 std::string word);
 75  bool horizontal (std::vector<std::vector<char>> cw, short row, short colum
 76                   std::string word);
 77  bool diagonal (std::vector<std::vector<char>> cw, short row, short column,
 78                 std::string word);
```

```
    79
    80  bool fit_vertical(std::vector<std::vector<char>> cw, short row,
    81                    std::string word);
    82  bool fit_horizontal(std::vector<std::vector<char>> cw, short column,
    83                      std::string word);
    84  bool fit_diagonal(std::vector<std::vector<char>> cw,
    85                    short row, short column, std::string word);
    86
    87  bool vertical_match(std::vector<std::vector<char>> cw, short row, short col
    88                      std::string word);
    89  bool horizontal_match(std::vector<std::vector<char>> cw, short row, short c
    90                      std::string word);
    91  bool diagonal_match(std::vector<std::vector<char>> cw, short row, short col
    92                      std::string word);
    93
    94  std::vector<std::vector<char>> make_vertical( std::vector<std::vector<char>
    95                                                short row, short column,
    96                                                std::string word);
    97  std::vector<std::vector<char>> make_horizontal (std::vector<std::vector<cha
    98                                                  short row, short column,
    99                                                  std::string word);
   100  std::vector<std::vector<char>> make_diagonal (std::vector<std::vector<char>
   101                                                short row,short column,
   102                                                std::string word);
   103
   104
   105  std::vector<std::vector<char>> place_word(std::vector<std::vector<char>> cw
   106                                            std::string word);
   107
   108  std::vector<std::vector<std::vector<char>>> input_word(std::vector<std::vec
   109                                                         std::string s);
   110
   111
   112  #endif
bj94684@ares:~$ show-code crossword.cpp


crossword.cpp:


     1  #include<iostream>
     2  #include<string>
     3  #include<vector>
     4  #include<ctime>
     5  #include<cstdlib>
     6  #include<cctype>
     7  #include"crossword.h"
     8
     9  using namespace std;
    10
    11
    12  vector<vector<char>> fill_cw_w_l(vector<vector<char>> cw, char t)
    13  {
    14      vector<vector<char>>cw1 = cw;
    15      for (auto & r : cw1)
    16      {
    17          for (auto & c : r)
    18          {
    19              if (c == t)
    20              {
    21                  c = static_cast<char> (rand() % ( static_cast<short>('z') -
    22                                            static_cast<short>('a') + 1 )
    23                                          + static_cast<short>('a'));
    24              }
    25          }
    26      }
    27      return cw1;
    28  }
    29
    30  vector<vector<char>> gen_cw_ans(vector<vector<char>> cw, char t)
    31  {
    32      vector<vector<char>>cw1 = cw;
    33      for (auto & r : cw1)
    34      {
    35          for (auto & c : r)
    36          {
    37              if (c != t)
    38              {
    39                  c = static_cast<char>(toupper(c));
    40              }
    41          }
    42      }
    43      return cw1;
    44  }
    45
    46
    47
    48  bool vertical(vector<vector<char>> cw, short row, short column, string word
    49  {
    50      bool in;
    51      short max_rows = static_cast<short>(cw.size() - 1);
    52      short max_col = static_cast<short>(cw[0].size( ) - 1);
    53      short r_loc = 0;
    54      short c_loc = 0;
    55      short num = 0;
    56      for (string::size_type r = 0;
    57           r != word.length(); ++r)
    58      {
    59          r_loc = row + static_cast<short>(r);
    60          c_loc = column;
    61          if (r_loc <= max_rows && c_loc <= max_col)
    62          {
    63              if (cw[row + r][column] == word[r] || cw[row + r][column] == '
    64              {
    65                  ++num;
    66              }
    67          }
    68      }
```

```
    if (num == static_cast<short>( word.length() ) )
    {
        return in = true;
    }
    return in = false;
}
bool horizontal(vector<vector<char>> cw, short row, short column, string w
{
    bool in;
    short num = 0;
    short max_rows = static_cast<short>(cw.size() - 1);
    short max_col = static_cast<short>(cw[0].size( ) - 1);
    short r_loc = 0;
    short c_loc = 0;
    for (string::size_type r = 0;
         r != word.length(); ++r)
    {
        r_loc = row;
        c_loc = column + static_cast<short>(r);
        if (r_loc <= max_rows && c_loc <= max_col)
        {
            if (cw[row][column + r] == word[r] || cw[row][column + r] == '
            {
                ++num;
            }
        }
    }
    if (num == static_cast<short>( word.length() ) )
    {
        return in = true;
    }
    return in = false;
}
bool diagonal(vector<vector<char>> cw, short row, short column, string wor
{
    bool in;
    short num = 0;
    short max_rows = static_cast<short>(cw.size() - 1);
    short max_col = static_cast<short>(cw[0].size( ) - 1);
    short r_loc = 0;
    short c_loc = 0;
    for (string::size_type r = 0;
         r != word.length(); ++r)
    {
        r_loc = row + static_cast<short>(r);
        c_loc = column + static_cast<short>(r);
        if (r_loc <= max_rows && c_loc <= max_col)
        {
            if (cw[row + r][column + r] == word[r] || cw[row + r][column +
            {
                ++num;
            }
        }
    }
```

```
    if (num == static_cast<short>( word.length()) )
    {
        return in = true;
    }
    return in = false;

}

bool fit_vertical(vector<vector<char>> cw, short row, string word)
{
    bool in;
    short num = 0;
    short row_limit = static_cast<short>(cw.size() );
    //short col_limit = static_cast<short>(cw[0].size() );
    for (string::size_type r = 0;
         r != word.length(); ++r)
    {
        num = row + static_cast<short>(r);
        if (  num > row_limit)
        {
            return in = false;
        }
    }
    return in = true;

}
bool fit_horizontal(vector<vector<char>> cw, short column, string word)
{
    bool in;
    short num = 0;
    //short row_limit = static_cast<short>(cw.size() );
    short col_limit = static_cast<short>(cw[0].size() );
    for (string::size_type r = 0;
         r != word.length(); ++r)
    {
        num = column + static_cast<short>(r);
        if (  num > col_limit )
        {
            return in = false;
        }
    }
    return in = true;

}
bool fit_diagonal(vector<vector<char>> cw, short row, short column, string

{
    bool in;
    short num = 0;
    short num1 = 0;
    short row_limit = static_cast<short>(cw.size());
    short col_limit = static_cast<short>(cw[0].size());
    for (string::size_type r = 0;
         r != word.length(); ++r)
```

```
    {
        num = row + static_cast<short>(r);
        num1 = column + static_cast<short>(r);
        if (  num > row_limit || num1 > col_limit )
        {
            return in = false;
        }
    }
    return in = true;
}

bool vertical_match(vector<vector<char>> cw, short row, short column, strin
{
    bool in;
    short max_rows = static_cast<short>(cw.size() - 1);
    short max_col = static_cast<short>(cw[0].size( ) - 1);
    short r_loc = 0;
    short c_loc = 0;
    short num = 0;
    short num1 = 0;
    for (string::size_type r = 0;
        r != word.length(); ++r)
    {
        r_loc = row + static_cast<short>(r);
        c_loc = column;
        if (r_loc <= max_rows && c_loc <= max_col)
        {
            if (cw[row + r][column] == word[r] || cw[row + r][column] == '
            {
                ++num;
            }
            if (cw[row + r][column] == word[r])
            {
                ++num1;
            }
        }
    }
    if (num == static_cast<short>( word.length() ) && num1 != 0)
    {
        return in = true;
    }
    return in = false;
}
bool horizontal_match(vector<vector<char>> cw, short row, short column, st
{
    bool in;
    short max_rows = static_cast<short>(cw.size() - 1);
    short max_col = static_cast<short>(cw[0].size( ) - 1);
    short r_loc = 0;
    short c_loc = 0;
    short num = 0;
    short num1 = 0;
    for (string::size_type r = 0;
        r != word.length(); ++r)
```

```
    {
        r_loc = row ;
        c_loc = column + static_cast<short>(r);
        if (r_loc <= max_rows && c_loc <= max_col)
        {
            if (cw[row][column + r] == word[r] || cw[row][column + r] == '
            {
                ++num;
            }
            if (cw[row][column + r] == word[r])
            {
                ++num1;
            }
        }
    }
    if (num == static_cast<short>( word.length() ) && num1 != 0)
    {
        return in = true;
    }
    return in = false;
}
bool diagonal_match(vector<vector<char>> cw, short row, short column, strin
{
    bool in;
    short max_rows = static_cast<short>(cw.size() - 1);
    short max_col = static_cast<short>(cw[0].size( ) - 1);
    short r_loc = 0;
    short c_loc = 0;
    short num = 0;
    short num1 = 0;
    for (string::size_type r = 0;
        r != word.length(); ++r)
    {
        r_loc = row + static_cast<short>(r);
        c_loc = column + static_cast<short>(r);
        if (r_loc <= max_rows && c_loc <= max_col)
        {
            if (cw[row + r][column + r] == word[r] || cw[row + r][column +
            {
                ++num;
            }
            if (cw[row + r][column + r] == word[r])
            {
                ++num1;
            }
        }
    }
    if (num == static_cast<short>( word.length() ) && num1 != 0)
    {
        return in = true;
    }
    return in = false;

}
```

```
285
286 vector<vector<char>> make_vertical (vector<vector<char>> cw, short row,
287                     short column, string word)
288 {
289     vector<vector<char>>cw1 = cw;
290     for (string::size_type r = 0;
291          r != word.length(); ++r)
292     {
293         cw1[row + r][column] = word[r];
294
295     }
296     return cw1;
297 }
298
299 vector<vector<char>> make_horizontal (vector<vector<char>> cw, short row,
300                     short column, string word)
301 {
302     vector<vector<char>>cw1 = cw;
303     for (string::size_type r = 0;
304          r != word.length(); ++r)
305     {
306         cw1[row][column + r] = word[r];
307
308     }
309     return cw1;
310 }
311 vector<vector<char>> make_diagonal (vector<vector<char>> cw, short row,
312                     short column, string word)
313 {
314     vector<vector<char>>cw1 = cw;
315     for (string::size_type r = 0; r != word.length(); ++r)
316     {
317         cw1[row + r][column + r] = word[r];
318     }
319     return cw1;
320 }
321
322 vector<vector<char>> place_word(vector<vector<char>> cw, string word)
323 {
324     vector<vector<char>> cw1 = cw;
325     bool done;
326     do{
327         short row = rand_num(0,9);
328         short col = rand_num(0,9);
329         short indicator = rand_num(0,1);
330         short indicator2 =rand_num(1,3);
331         if (indicator == 1)
332         {
333             word = reverse_word(word);
334         }
335         if(indicator2 == 1)
336         {
337             if (fit_vertical(cw1, row, word))
338             {
339                 if (vertical(cw1, row, col, word) )
340                 {
341                     cw1 = make_vertical(cw1, row, col, word);
342                     done = true;
343                 }
344                 else
345                 {
346                     done = false;
347                 }
348             }
349             else
350             {
351                 done = false;
352             }
353         }
354         else if (indicator2 == 2)
355         {
356             if (fit_horizontal(cw1,col, word))
357             {
358                 if (horizontal(cw1, row, col, word) )
359                 {
360                     cw1 = make_horizontal(cw1, row, col, word);
361                     done = true;
362                 }
363                 else
364                 {
365                     done = false;
366                 }
367             }
368             else
369             {
370                 done = false;
371             }
372         }
373         else
374         {
375             if (fit_diagonal(cw1, row, col, word))
376             {
377                 if (diagonal(cw1, row, col, word) )
378                 {
379                     cw1 = make_diagonal(cw1, row, col, word);
380                     done = true;
381                 }
382                 else
383                 {
384                     done = false;
385                 }
386             }
387             else
388             {
389                 done = false;
390             }
391         }
392     }while( ! done);
```

```
393     return cw1;
394 }
395
396 vector < vector< vector<char> > > input_word(vector<vector<char>> cw, stri
397 {
398     vector < vector< vector<char> > > crossword;
399     vector<vector<char>> cw1 = cw;
400     vector<vector<char>> cw12 = cw;
401     for (vector<vector<char> >::size_type row = 0;
402          row != cw1.size(); ++row)
403     {
404         for (vector<char>::size_type col = 0;
405             col != cw1[row].size(); ++col)
406         {
407                 if (vertical_match(cw1, row, col, s) )
408                 {
409                     cw12 = make_vertical(cw1, row, col, s);
410                     crossword.push_back(cw12);
411                 }
412         }
413     }
414     vector<vector<char>> cw2 = cw;
415     vector<vector<char>> cw22 = cw;
416     for (vector<vector<char> >::size_type row = 0;
417          row != cw2.size(); ++row)
418     {
419         for (vector<char>::size_type col = 0;
420             col != cw2[row].size(); ++col)
421         {
422                 if (horizontal_match(cw2, row, col, s) )
423                 {
424                     cw22 = make_horizontal(cw2, row, col, s);
425                     crossword.push_back(cw22);
426                 }
427         }
428     }
429     vector<vector<char>> cw3 = cw;
430     vector<vector<char>> cw32 = cw;
431     for (vector<vector<char> >::size_type row = 0;
432          row != cw3.size(); ++row)
433     {
434         for (vector<char>::size_type col = 0;
435             col != cw3[row].size(); ++col)
436         {
437                 if (diagonal_match(cw3, row, col, s) )
438                 {
439                     cw32 = make_diagonal(cw3, row, col, s);
440                     crossword.push_back(cw32);
441                 }
442         }
443     }
444
445     return crossword;
446 }
```

```
bj94684@ares:~$ show-code cw.cpp


cw.cpp:


 1  #include<iostream>
 2  #include<fstream>
 3  #include<string>
 4  #include<vector>
 5  #include<ctime>
 6  #include<cctype>
 7  #include"crossword.h"
 8
 9  using namespace std;
10
11  inline vector<string>::size_type maximum(const vector<string> & vec)
12        {
13            vector<string>::size_type max = 0;
14            for (vector<string>::size_type at = 1;
15                 at < vec.size(); ++at)
16            {
17                if (vec[at].length() > vec[max].length())
18                {
19                  max = at;
20                }
21           }
22            return max;
23       }
24
25
26  int main()
27  {
28      srand(static_cast<unsigned>(time(nullptr)));
29      cout << "\n\t\tWelcome to the Crossword Generator\n\n";
30      ifstream file;
31      string fn;
32      cout << "\nPlease enter the name of your names files:  ";
33      getline(cin, fn);
34      file.open(fn);
35      while ( ! file )
36      {
37          file.close();
38          file.clear();
39          cout << "\nIm sorry I could open " << '"' << fn << '"'
40               << ". Please enter another name:  ";
41          getline(cin, fn);
42          file.open(fn);
43      }
44      cout << "\nFile " << '"' << fn << '"' << " was opened succesfully\n";
45      string s;
46      vector<string> names;
47      file >> ws;
48      while (!file.eof())
```

```
    {
        file >> s;
        names.push_back(s);
        file >> ws;
    }
    file.close();
    file.clear();
    vector<string>::size_type max_loc = maximum(names);
    short max_size = static_cast<short>(names[max_loc].length( ) + 2);
    string word1 = names[0];
    //cout << "The Number of Words are: " << names.size() << '\n';
    //cout << "The Biggest word is the " << max_loc << " of the list\n";
    vector<vector<char>> crossword1;
    crossword1 = generate_cw(crossword1,max_size,max_size);
    vector<vector<char>>crossword = fill_cw(crossword1, '*');
    vector<vector<char> > cw1;
    short indicator;
    long tries = 0;
    long max_tries = 50;
    do
    {
        indicator = 1;
        vector<vector<char> > cw = place_word(crossword, word1);
        cw1 = cw;
        vector < vector<vector<char>> >cw_u;
        for (vector<string>::size_type r= 1;
            r != names.size(); ++r)
        {
            short reverse = rand_num(0,1);
            string n = names[r];
            if ( reverse == 1)
            {
                n = reverse_word(n);
            }
            cw_u = input_word(cw1, n);
            if ( ! cw_u.empty())
            {
                short i = rand_num(0,static_cast<short>(cw_u.size() - 1));
                cw1 = cw_u[i];
                ++indicator;
            }
            else
            {
                cw1 = place_word(cw1,n);
                if ( ! cw1.empty())
                {
                    ++indicator;
                }
                else
                {
                    break;
                }
            }
        }
```

```
        ++tries;
        //display(cw1);
        //cout <<'\n';
    }while ( indicator !=  static_cast<short>(names.size() ) || tries != ma
    //display(cw1);
    //cout << '\n';
    vector<vector<char> > cw_ans = gen_cw_ans(cw1,'*');
    vector<vector<char> > crossword_u = fill_cw_w_l(cw1,'*');
    cout << "The Generated Crossword:\n";
    display(crossword_u);
    cout <<"\nThe Answer Key:\n";
    display(cw_ans);
    cout << '\n';

    return 0;
}
```

```
bj94684@ares:~$ CPP crossword cw
crossword.cpp...
cw.cpp***
crossword.cpp: In function
‘std::vector<std::vector<std::vector<char> > >
input_word(std::vector<std::vector<char> >, std::string)’:
crossword.cpp:407:41: warning:
conversion from ‘std::vector<std::vector<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘short int’ may change value
[-Wconversion]
  407 |                 if (vertical_match(cw1, row,
  col, s) )
      |                                         ^~~
crossword.cpp:407:46: warning:
conversion from ‘std::vector<char>::size_type’
{aka ‘long unsigned int’} to ‘short
int’ may change value [-Wconversion]
  407 |                 if (vertical_match(cw1, row,
  col, s) )
      |
      ^~~
crossword.cpp:409:47: warning:
conversion from ‘std::vector<std::vector<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘short int’ may change value
[-Wconversion]
  409 |                     cw12 = make_vertical(cw1,
  row, col, s);
      |
      ^~~
crossword.cpp:409:52: warning:
conversion from ‘std::vector<char>::size_type’
{aka ‘long unsigned int’} to ‘short
int’ may change value [-Wconversion]
  409 |                     cw12 = make_vertical(cw1, row,
  col, s);
      |
```

```
      ^~~
crossword.cpp:422:43: warning:
conversion from ‘std::vector<std::vector<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘short int’ may change value
[-Wconversion]
  422 |                 if (horizontal_match(cw2, row,
  col, s) )
      |                                           ^~~
crossword.cpp:422:48: warning:
conversion from ‘std::vector<char>::size_type’
{aka ‘long unsigned int’} to ‘short
int’ may change value [-Wconversion]
  422 |                 if (horizontal_match(cw2, row,
  col, s) )
      |
      ^~~
crossword.cpp:424:49: warning:
conversion from ‘std::vector<std::vector<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘short int’ may change value
[-Wconversion]
  424 |                     cw22 = make_horizontal(cw2,
  row, col, s);
      |
      ^~~
crossword.cpp:424:54: warning:
conversion from ‘std::vector<char>::size_type’
{aka ‘long unsigned int’} to ‘short
int’ may change value [-Wconversion]
  424 |                     cw22 = make_horizontal(cw2, row,
  col, s);
      |
      ^~~
crossword.cpp:437:41: warning:
conversion from ‘std::vector<std::vector<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘short int’ may change value
[-Wconversion]
  437 |                 if (diagonal_match(cw3, row,
  col, s) )
      |                                         ^~~
crossword.cpp:437:46: warning:
conversion from ‘std::vector<char>::size_type’
{aka ‘long unsigned int’} to ‘short
int’ may change value [-Wconversion]
  437 |                 if (diagonal_match(cw3, row,
  col, s) )
      |
      ^~~
crossword.cpp:439:47: warning:
conversion from ‘std::vector<std::vector<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘short int’ may change value
[-Wconversion]
  439 |                     cw32 = make_diagonal(cw3,
  row, col, s);
      |
      ^~~
crossword.cpp:439:52: warning:
conversion from ‘std::vector<char>::size_type’
{aka ‘long unsigned int’} to ‘short
int’ may change value [-Wconversion]
  439 |                     cw32 = make_diagonal(cw3, row,
  col, s);
      |
      ^~~


bj94684@ares:~$ ./cw.out

                Welcome to the Crossword Generator


Please enter the name of your names files:  sjsdsa

Im sorry I could open "sjsdsa". Please enter another name:  cw_names1

File "cw_names1" was opened succesfully
The Generated Crossword:
l e v d y o f x f b
e r q g b p o s i c
h t i j m y s x g j
t j p r o p e r t y
v x f a u v t f u a
i y r z r h s q x m
b z t w n j d f u b
e a l x i l e a k m
i u r i n p h t y m
m q k m g a k f b r

The Answer Key:
* * * * * * * * * *
* * * * * * * * * *
* * * * M * * * * *
* * P R O P E R T Y
* * * * U * * * * *
* * * * R * * * * *
* * * * N * D * * *
* * * * I L E A K *
* * * * N * H T Y M
* * * * G * * * * *

bj94684@ares:~$ ./cw.out

                Welcome to the Crossword Generator
```

```
Please enter the name of your names files:  cw_names2

File "cw_names2" was opened succesfully
The Generated Crossword:
g r r u h o c y z p u i u
m n n s r l i m s c q e c
e i o v s o k g x b a r k
q v i i t p l f z h r o b
l z t i a k x r o n y y r
r g a m n e i q s h d t u
y e r k d v x u o b a h t
a o e d e l i c a t e n r
p y p l a n e s e q r b v
k q o k i h b o i e v l g
p s o b k t h u p b d y y
w t c y m g s w e l l r d
g i f d u x n c h o k e c

The Answer Key:
* * * * * * * * * * * *
* * N * * * * * * * * *
* * O * S * * * * * * *
* * I * T * * * * * * *
* * T I A * * * * * Y * *
* * A * N E * * * * D * *
* * R * D V X * * * A * *
* * E D E L I C A T E * *
* * P L A N E S E * R * *
* * O * * * * * I E * * *
* * O * * * * * * B D * *
* * C * * * S W E L L * *
* * * * * * * C H O K E *

bj94684@ares:~$ ./cw.out

                Welcome to the Crossword Generator


Please enter the name of your names files:  cw_names1

File "cw_names1" was opened succesfully
The Generated Crossword:
p p d a y q o a j i
w y k j f a u d n h
y t r e p o r p g d
v m w c r o d r o b
h z s z s g z b p x
b b m o u r n i n g
g i b y c n p j w u
h k s j t b g o v x
q e y i d h q r e t
k a e l i c w j k h

The Answer Key:
* * D A Y * * * * *
* * * * * * * * * *
Y T R E P O R P * *
* * * * * * * * * *
* * * * * * * * * *
* * M O U R N I N G
* * * Y * * * * * *
* * * * T * * * * *
* * * * * H * * * *
K A E L * * * * * *

bj94684@ares:~$ ./cw.out

                Welcome to the Crossword Generator


Please enter the name of your names files:  cw_names1

File "cw_names1" was opened succesfully
The Generated Crossword:
j c b l z n f g g m
m j b b e m f n f d
v y q r p a o i w s
d y s b b r k n a k
c a e n b u v r k i
h d t j k h o u s k
k q y t r e p o r p
a d h o h t y m z z
t u c v l e h n j u
p f p z n y u w l b

The Answer Key:
* * * L * * * G * *
* * * * E * * N * *
* * * * * A * I * *
* Y * * * * K N * *
* A * * * * * R * *
* D * * * * * U * *
* * Y T R E P O R P
* * * * H T Y M * *
* * * * * * * * * *
* * * * * * * * * *

bj94684@ares:~$ ./cw.out

                Welcome to the Crossword Generator


Please enter the name of your names files:  cw_names2

File "cw_names2" was opened succesfully
The Generated Crossword:
b e z z a y p h p l y h a
m k n e t a c i l e d i i
```

```
g o e x p l a n e k a t g
b f h k i e o v w o e d q
b e p s b x d i g h r k u
r u w g w c e s l c o n r
f t p a d e z i q f j c p
x y s p n e l b b z g e e
p r y g a d l l w o r k f
g n o i t a r e p o o c b
k z s k s b k r m o h j p
b b q a j x e y q e h d d
v u n c d q m c j y f v s

The Answer Key:
* * * * * * * * * * Y * *
* * * E T A C I L E D * *
* * * * P L A N E K A * *
* * * * * E * V * O E * *
* * * S * X * I * H R * *
* * * * W C * S * C * * *
* * * * D E * I * * * * *
* * * * N E L B * * * * *
* * * * A D * L * * * * *
* N O I T A R E P O O C *
* * * * S * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * *

bj94684@ares:~$ exit
exit

Script done on 2023-07-13 22:13:52-05:00 [COMMAND_EXIT_CODE="0"]
```