```
/* encrypt.c  Program to encrypt text elements of a data file
 * before writing and re assemble after reading. I'm trying to create
 * what I had written in a pascal program many years ago but no longer
 * have the code for. Program date 06/25/2025 updated 09/29/2025 by Ronald Lewis
 * This is the final build for now */

 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>

struct account *start; //pointer to first entry in list
struct account *last;  //pointer to the last entry
struct Eaccount *start_e; //pointer to first entry in list
struct Eaccount *last_e;  //pointer to the last entry



struct Eaccount *encode_node();
struct Eaccount *find_duplicate( char *);

void list( void );
void find_name( void );
void addChar(char *s, char c);
void enter(void);
void write_list(void);
void save_as_encoded( void );
void load_normal( void );
void load_encoded( void );
void decode_node(char *n, int code);
void deleteSingleItem( struct Eaccount **start_e,  struct Eaccount **last_e );

void Make_Linked_List(struct account *i, struct account **start, struct account **last);
void Make_Encrypted_Linked_List(struct Eaccount *i_e, struct Eaccount **start_e, struct Eaccount
**last_e);



FILE *fp;  /* pointer to file to use */
int filefound = 0;
char TodaysDate[9] = "";  //used for creating files
char newstring[30] = "";
const int ARRAY_LENGTH = 93;
```

```
int match = 0; // used in find_duplicate() function and in enter()


/* template holds the printable characters of 33 - 126 used to replace characters read.
 * from text. No unprintable, no sounds, or spec chars */

/* first element is the key and second element is the replacement character
 * here in each of the 5 arrays, you will place random character codes, beginning with
 * character 33 - 126.  Each of the 5 should be different than the others.  There should
 * be a space between each of the numbers.  This is for your own encryption.  */

char template[5][94] = { {   }, {   }, {   }, {  }, {   } };

/* structure of an individual node in the LL*/
 struct Eaccount {
       char Eaccount_name[30]; // name of the account, business, store, etc.
    char Eaccount_web_address[30]; // website address associated with account
    char Euser_name[30]; // the user name for this account (usually not the full name)
    char Epassword[30]; // the password for this account
    char Eemail[30]; // user email address used for this account
    int  Eecode; // encryption key code for this node
    char Ecomments[30]; // users comments for this account, 2fa, email verification, etc.
    struct Eaccount *prior; //  pointers to the previous and next items
    struct Eaccount *next;
} Elist;


void main() {

  char buf[30] = "";

/* initialize top and bottom pointers */
  start = last = NULL;
/* clear the screen and start with 3 lines down from top spacing */
  clrscr();
  printf("\n\n\n");
/* if file exists load it, otherwise skip to menu */
   fp = fopen( "readt_e", "rb" );
        if( !fp ) {
             printf( "              readt_e not found - continue to menu\n" ); getch(); }
              else { fclose(fp); load_encoded(); filefound = 1; }
```

```
/* enter todays date...used for filenames */
 for( ; ; )  {
   clrscr();
   printf("\n\n                Enter Todays date: yyyymmdd:   ");
   scanf("%s", TodaysDate);
   printf("\n\n        To confirm, please re enter Todays date: yyyymmdd:   ");
   scanf("%s", buf);
   if(strcmp(TodaysDate, buf) != 0)  {
    clrscr();
    printf("               Dates do not match - press any key to continue\n  "); getch(); }
     else break;  }

/* start the menu */
 for( ; ; )  {
       switch(menu_select())  {
      case 1: enter();
           break;
         case 2: deleteSingleItem( &start_e, &last_e );
           break;
         case 3: list();
           break;
         case 4: find_name();
           break;
         case 5: write_list();
           break;
      case 6: save_as_encoded();
           break;
      case 7: exit(0);
         case 8: save_as_encoded(); exit(0);

       }//switch
      }//for

   return;
} //main

/* select an operation */
menu_select()  {
 char s[80];
 int c;
 clrscr();
 printf("\n              Password manager tool       Updated: 20250929Final");
```

```
 if ( filefound == 1 ) printf("\n          readt_e loaded");
 if ( filefound == 0 ) printf("\n          readt_e not loaded, press 8 when finished adding nodes.");
 printf("\n\n\n");
 printf("              1. Enter a new account\n");
 printf("              2. Delete an account\n");
 printf("              3. List all current accounts\n");
 printf("              4. Search an account by its name\n");
 printf("              5. Write the list of accounts to a file\n");
 printf("              6. Save encoded readt_e file\n");
 printf("              7. Exit program WITHOUT saving\n");
 printf("              8. Save encoded readt_e file and EXIT\n");
  do {
  printf("\n          Enter your choice:   ");
  gets(s);
  c = atoi(s);
 }//do
 while ( c < 0 || c > 8 );
 return c;
}//menu_select function


/* find a grocery Item by its name */
void find_name() {
      struct Eaccount *info_e;
      int count = 0;
      int found = 0;
      int total_found = 0;

      char name[30];
      char buffer[30];
      char n[30];
      int code=1;

      printf("\n          Enter the item name to search:  ");
      scanf("%s", buffer);
      strcpy( name, buffer );

   info_e = start_e;

   printf("\n"); // skips a line first

   while( info_e ) {
```

```
if(strcmp( name, info_e->Eaccount_name ) == 0 )  { code = info_e->Eecode;
            strcpy(n, info_e->Eemail);
            clrscr();
            printf("                    Searching for: \"%s\"\n", name);
            decode_node(n, code);
          printf("\n\n   Decoded...\n");
          printf("   My email: %s\n", newstring);
          strcpy(newstring, "");
         }

     if(strcmp( name, info_e->Eaccount_name ) == 0 )  { found = 1; code = info_e->Eecode;
   strcpy(n, info_e->Euser_name);
          decode_node(n, code);
          printf("   Username: %s\n", newstring);
          strcpy(newstring, "");
         }

     if(strcmp( name, info_e->Eaccount_name ) == 0 )  { code = info_e->Eecode;
            strcpy(n, info_e->Epassword);
       decode_node(n, code);
       printf("   Password: %s\n", newstring);
       strcpy(newstring, "");
       }

     if(strcmp( name, info_e->Eaccount_name ) == 0 )  { code = info_e->Eecode;
            strcpy(n, info_e->Eaccount_web_address);
       decode_node(n, code);
       printf("   Website: %s\n", newstring);
       strcpy(newstring, "");
       }

     if(strcmp( name, info_e->Eaccount_name ) == 0 )  { code = info_e->Eecode;
            strcpy(n, info_e->Ecomments);
       decode_node(n, code);
       printf("   Other info: %s\n", newstring);
       strcpy(newstring, "");
       }

     if( found > 0) { total_found += 1; }

     info_e = info_e->next;
```

```
          /* reset the found flag to 0 */
          found = 0;
          count += 1;

        } //end while

       /* print the number of items in list minus the head and tail nodes */
    if(( total_found > 0 ) && ( count > 2 )) printf("\n\n       Searched %d items for the item name and
found %d item matching", ( count ), total_found );
    printf("\n\n");
       if( total_found == 0 ) printf("              nothing found " );

       getch();
       return NULL;
      }  //end find_name

void decode_node(char *n, int code) {
   char newch;
      int  value;
      int  index1 = 0;

   /* unencrypt each character in string n[] */
   for( index1 = 0; index1 < strlen(n); index1++ ) {
    value = n[index1] - 33;
    newch = template[code][value];
    addChar(newstring, (char)newch );
   }
}


/* load encrypted file to list */
 void load_encoded( void )  {
       struct Eaccount *info_e;

       fp = fopen( "readt_e", "rb" );
       if( !fp ) {
            printf( "cannot open read_e\n" );
            getch();
            exit( 1 );
             }
       /* free any previously allocated memory */
       while( start_e )  {
```

```
            info_e = start_e->next;
            free( info_e );
            start_e = info_e;
            }
        /* reset top and bottom pointers */
        start_e = last = NULL;

        while( !feof( fp ) )  {
            info_e = ( struct Eaccount * ) malloc( sizeof( struct Eaccount ) );
            if( !info_e )  {
                printf( "out of memory" );
                return;
                }
           if( 1 != fread( info_e, sizeof( struct Eaccount ), 1, fp ) )  { break; }

           Make_Encrypted_Linked_List( info_e, &start_e, &last_e );
            } //end while
        fclose( fp );
    }  // end load


/* save list to disk file main and backup */
void save_as_encoded( void ) {
        struct Eaccount *info_e;
        FILE *fp1;
        FILE *fp2;

        fp1 = fopen( "readt_e",  "wb" );
        if( !fp1 ) {
            printf( "cannot open readt_e\n" );
            exit( 1 );
            }

        fp2 = fopen( "readt_e.bak",  "wb" );
        if( !fp2 ) {
            printf( "cannot open readt_e.bak\n" );
            exit( 1 );
            }

        info_e = start_e;
        while( info_e ) {
            fwrite( info_e, sizeof( struct Eaccount ), 1, fp1 );
```

```
        info_e = info_e->next;
        }
    fclose( fp1 );
    printf( "\n        saved readt_e" );

    info_e = start_e;
    while( info_e ) {
        fwrite( info_e, sizeof( struct Eaccount ), 1, fp2 );
        info_e = info_e->next;
        }
    fclose( fp2 );

    printf( "\n        saved readt_e.bak .. press a key to continue" );
    getch();
    }


/* Create a doubly linked list in sorted order */
void Make_Encrypted_Linked_List( struct Eaccount *i_e,    //new element
  struct Eaccount **start_e,              //first element in the list
  struct Eaccount **last_e )              //last element in the list
  {
        struct Eaccount *old_e, *p_e;

        if(*last_e == NULL)  {            //first element in the list
         i_e->next = NULL;
         i_e->prior = NULL;
         *last_e = i_e;
         *start_e = i_e;
         return;
         }

 p_e = *start_e;                          //start at top of the list
 old_e = NULL;

 while(p_e)  {
  if(strcmp(p_e->Eaccount_name, i_e->Eaccount_name) < 0)  {
      old_e = p_e;
      p_e = p_e->next;
    }
    else  {
     if(p_e->prior)  {
```

```
          p_e->prior->next = i_e;
          i_e->next = p_e;
          i_e->prior = p_e->prior;
          p_e->prior = i_e;
          return;
        }
      i_e->next = p_e;                    //new first element
      i_e->prior = NULL;
      p_e->prior = i_e;
      *start_e = i_e;
      return;
      }  //end else

  }  //end while
  old_e->next = i_e;                    //put on end
  i_e->next = NULL;
  i_e->prior = old_e;
  *last_e = i_e;

} //end Make_Linked_List function


/* find an item by its name */
struct Eaccount *find_duplicate( char *buffer ) {
      struct Eaccount *info_e;

      info_e = start_e;

   printf("\n"); // skips a line first

   while( info_e ) {
    if(strcmp( buffer, info_e->Eaccount_name ) == 0 )  { match = 1;
              printf("          %s is already used, returning to menu\n", info_e->Eaccount_name);
              getch(); break; }

            else info_e = info_e->next;
        }

       return NULL;
      }  //end find_duplicate
```

```
/* write the entire list to a text file */
void write_list(void) {
      struct Eaccount *info_e;

      //char TodaysDate[9] = "";  //used for creating files
      char buffer_previousName[30] ="empty";
      char buffer[30];
      char buf[9];     //for storing date string
      char City[30];
      int count = 0;    //counts the nodes except the head and tail node

   /* make a unique file name for the file */
      char file_extension[4] = ".lst";
      char newfilename[30]="";

      FILE *text_listing;

      /* set info pointer to beginning of the encrypted LL*/
      info_e = start_e;

      /* clear old file or create a new clean file */
   strcpy(newfilename, TodaysDate);
   strncat(newfilename, file_extension, 4);

      text_listing=fopen(newfilename,"w");

      clrscr();
      printf("              Printing the list of items ...\n\n ");  //print this to the screen

      /* begin writing search results to the file */
      fprintf(text_listing, "                                  Accounts as of: %s\n\n", TodaysDate);
      while(info_e)  {
            if( strcmp( info_e->Eaccount_name, "000000" ) != 0  && strcmp( info_e->Eaccount_name,
"zzzzzz" ) != 0 ) {
               if( strcmp( info_e->Eaccount_name, buffer_previousName) != 0 )  {
            /* print the item and then a line for writing notes */
            fprintf(text_listing,"  %s  \n\n\n", info_e->Eaccount_name );

            /* increment count for each item found */
            /* get the next item */
            count +=1; } }
```

```
            strcpy(buffer_previousName, info_e->Eaccount_name );

            info_e = info_e->next;
         }

    /* this prints to .lst file */
    fprintf(text_listing,"  %d items are in this list...", count );

    fclose(text_listing);

        /* this prints to the screen */
        printf("      The text file: %s  has been created \n       with %d items - press any key", newfilename,
count );
        getch();
  } //end of write_list()


/* remove an element from the list */
void deleteSingleItem( struct Eaccount **start_e,  struct Eaccount **last_e )  {
        struct Eaccount *info_e, *find_single_item();
        char acctName[30];
    char buffer[30];

        printf("\n             Enter user account name:  ");
        scanf("%s", buffer);
        strcpy(acctName, buffer);

        info_e = find_single_item( acctName );

        if(info_e) {
            if(*start_e == info_e) {
                *start_e = info_e->next;
                if(*start_e)  (*start_e)->prior = NULL;
                else  *last_e = NULL;
                }
             else {
                 info_e->prior->next = info_e->next;
                 if(info_e != *last_e)  info_e->next->prior = info_e->prior;
                 else  *last_e = info_e->prior;
                 }
            printf("   DELETED - %s ", info_e->Eaccount_name);
            free(info_e);
```

```
            }
        getch();
        }  //end delete


/* find a single item to delete */
struct Eaccount *find_single_item( char *n ) {
        struct Eaccount *info_e;

    info_e = start_e;
        while( info_e ) {
            if (strcmp( n, info_e->Eaccount_name ) == 0 ) { return info_e; }
            info_e = info_e->next;
         }
         if(!info_e) printf("              nothing found " );
         return NULL;
        }  //end findname


/* enter new account name items  */
void enter()  {
  //initialize variables for encoding here
    char buffer1[30];
        char buffer2[30];
        char buffer3[30];
        char buffer4[30];
        char buffer5[30];
        char buffer6[30];
        int  count = 0;
        int  index1 = 0;
        int  index2 = 0;

  //char userEntry[30] = "";

  struct Eaccount *info_e;

  int encryption_code = 0;

  for( ; ; )  {
    info_e = ( struct Eaccount * )malloc(sizeof( Elist ) );
    if(!info_e)  {
         printf("\n out of memory");
```

```
 return;
 }//if(!info_e)

/* start inputing data to the node */
      clrscr();
      printf("\n\n");
      printf("\n   Enter the account or business name.\n");
   printf("   You must enter a string of at least one character here" );
   printf("\n   > ");
   scanf("%s", buffer1);

 /* if an account with this name already exists, don't allow */
   find_duplicate( buffer1 );
   /* if a duplicate name found then quit entry */
   if( match != 0 ) { strcpy(info_e->Eaccount_name, "quit"); match = 0; break; }
    else { strcpy(info_e->Eaccount_name, buffer1); }

 /* stop entry if Quit is typed in */
   if( strcmp(info_e->Eaccount_name, "Quit" ) == 0) break;
      if( strcmp(info_e->Eaccount_name, "quit" )  == 0) break;
      if( strcmp(info_e->Eaccount_name, "QUIT" )  == 0) break;

      printf("\n   Enter the user Name.\n");
   printf("   You must enter a string of at least one character here" );
   printf("\n   > ");
   scanf("%s", buffer2);

 /* choose the encryption code and use code 0 if entered incorrect */
   printf("\n   Enter eCode 0 - 4 for this node.   ");
   printf("\n   > ");
   scanf("%d", &encryption_code);
   if( (encryption_code > 4 ) || (encryption_code < 0 ) ) { encryption_code = 0; }
   info_e->Eecode = encryption_code;

   printf("\n   Enter the user password.\n");
   printf("   You must enter a string of at least one character here" );
   printf("\n   > ");
   scanf("%s", buffer3);

printf("\n   Enter the accounts website.\n");
   printf("   You must enter a string of at least one character here" );
   printf("\n   > ");
```

```
        scanf("%s", buffer4);

      printf("\n   Enter the email for user.\n");
        printf("   You must enter a string of at least one character here" );
        printf("\n   > ");
        scanf("%s", buffer5);

      printf("\n   Enter any additional comments.\n");
      printf("   You must enter a string of at least one character here" );
        printf("\n   > ");
        scanf("%s", buffer6);

/* ******************** begin encryption ******************** */

    /* encrypt each character info->user_name */
    strcpy(info_e->Euser_name, "");

    for( index1 = 0; index1 < strlen( buffer2 -1 ); index1++ ) {

     count = 0;
     for( index2 = 0; index2 <= ARRAY_LENGTH; index2++) {
      if( buffer2[index1] == template[info_e->Eecode][index2] ) {
         count = index2+33; // use the printable characters only
         addChar(info_e->Euser_name, (char)count );
         break;
       }
      }
     }

    /* encrypt each character info->password */
    strcpy(info_e->Epassword, "");

    for( index1 = 0; index1 < strlen( buffer3 - 1 ); index1++ ) {

     count = 0;
     for( index2 = 0; index2 <= ARRAY_LENGTH; index2++) {
      if( buffer3[index1] == template[info_e->Eecode][index2] ) {
         count = index2+33; // use the printable characters only
         addChar(info_e->Epassword, (char)count );
         break;
       }
      }
```

```
 }

/* encrypt each character info->account_web_address */
strcpy(info_e->Eaccount_web_address, "");

for( index1 = 0; index1 < strlen( buffer4 - 1 ); index1++ ) {

  count = 0;
  for( index2 = 0; index2 <= ARRAY_LENGTH; index2++) {
    if( buffer4[index1] == template[info_e->Eecode][index2] ) {
      count = index2+33; // use the printable characters only
      addChar(info_e->Eaccount_web_address, (char)count );
      break;
    }
  }
}

/* encrypt each character info->email */
strcpy(info_e->Eemail, "");

for( index1 = 0; index1 < strlen( buffer5 -1 ); index1++ ) {

  count = 0;
  for( index2 = 0; index2 <= ARRAY_LENGTH; index2++) {
    if( buffer5[index1] == template[info_e->Eecode][index2] ) {
      count = index2+33; // use the printable characters only
      addChar(info_e->Eemail, (char)count );
      break;
    }
  }
}

/* encrypt each character info->comments */
strcpy(info_e->Ecomments, "");

for( index1 = 0; index1 < strlen( buffer6 -1 ); index1++ ) {

  count = 0;
  for( index2 = 0; index2 <= ARRAY_LENGTH; index2++) {
    if( buffer6[index1] == template[info_e->Eecode][index2] ) {
      count = index2+33; // use the printable characters only
      addChar(info_e->Ecomments, (char)count );
```

```
        break;
       }
      }
     }

/* ****** end of encryption, add encrypted node to linked list ****** */

     Make_Encrypted_Linked_List(info_e, &start_e, &last_e);

   }//for loop for loading items
}//end of enter() function


void addChar(char *s, char c) {

   // Move pointer to the end
   while (*s++);

   // Append the new character
   *(s - 1) = c;

   // Add null terminator to mark new end
   *s = '\0';
}


/* display the entire list ... selection 3 */
void list() {
      struct Eaccount *info_e;
      int count = 0;
      int display_count = 0;

      info_e = start_e;
      clrscr();

      printf("    This is the entire listing of items in the file. \n");
      printf("    Today's date: %s ", TodaysDate);
      printf("\n\n\n\n");

      while(info_e)  {
            // the if statement prevents the dummy head and tail nodes from being printed
            if( strcmp( info_e->Eaccount_name, "000000" ) != 0  && strcmp( info_e->Eaccount_name,
```

```
"zzzzzz" ) != 0  ) {
            printf(" Account: %s\n username: %s \n password: %s   \n", info_e->Eaccount_name,
              info_e->Euser_name, info_e->Epassword );
            printf(" Website: %s  my email: %s \n", info_e->Eaccount_web_address, info_e->Eemail );
            printf(" comments for this account: %s \n\n", info_e->Ecomments );

          }
          /* increment count for each item found */
          /* get the next item */
          count +=1;  display_count += 1;
          //displays set number of items before continuing
          if(display_count == 3) { display_count = 0; getch(); }
          info_e = info_e->next;
      }
  /* print the number of items in list subtract the head and tail nodes
   * only prints if more than just head and tail nodes */

  if( count > 3 ) printf("\n  There are currently %d items in the password database.", ( count ));
  printf("\n\n");

    if(count == 0) printf("                nothing found " );
    getch();
    }  //end list
```