```
/* doubly linked list with menu for adding, deleting, saving,
 * items and reading items from a file. Programmed by Ron Lewis (Latest update: 20250604)
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "edays25.h"



/* structure of an individual node in the LL*/
struct groceryItem {
      char item_Name[30];  // the item name as one complete word (doesn't break rules for strings )
   float item_UnitCost;  // price divided by units ( ounces, ML, Lb., Pkg )
   char item_storeName[30];  // store where purchased
   char item_DatePurchased[9];  // sets the baseline to count days until expiration
   char item_City[30];  // the general city area where purchased
   int  item_days_until_expiration;  // sets the days to stay in list
   char item_Comments[30];    // how price was calculated
   int item_quality_rating;  // a rating to be used as a tiebreaker when prices are the same
   char item_Brand[30];  // the manufacturer name or brand name
   float item_actualprice_paid;  // the actual price paid for the item
   struct groceryItem *prior;  //  pointers to the previous and next items
   struct groceryItem *next;
} list_entry;



struct groceryItem *start; //pointer to first entry in list
struct groceryItem *last;  //pointer to the last entry
struct groceryItem *find_name( );
struct groceryItem *find_store( );
struct groceryItem *find_date( );
struct groceryItem *findbest();
struct groceryItem *find_single_item( char *, char *, char * );
struct groceryItem *remove_expired(struct groceryItem **, struct groceryItem **);

void load(void), list(void), write_list(void);
void save(void), saveDataBackup( void ), enter(void), modify(void);
void search4_item_expiration(void);
```

```
void deleteSingleItem(struct groceryItem **, struct groceryItem ** );
void deleteItems(struct groceryItem **, struct groceryItem **, struct groceryItem *info);
void Make_Linked_List(struct groceryItem *i, struct groceryItem **start, struct groceryItem **last);

/* set up an array to hold the DaysCounted values for each item */
int day[350];  //make sure the element number stays larger than the
            //number of items in the data list or pointers will crash

const int TRUE;
const int FALSE;

FILE *fp;
int expires = 0;
char search_name[30] = "empty";
int menu_select(void);
int submenu1(void);

void main( )  {
/* 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 "grocdat3" file exists load it, otherwise skip to menu */
   fp = fopen( "grocdat3", "rb" );
        if( !fp ) {
             printf( "list data not found - SKIPPING to menu\n" );
             } else { fclose( fp); load(); remove_expired( &start, &last );  }

/* start the menu */
  for( ; ; )  {
        switch(menu_select())  {
          case 1: findbest();
            break;
          case 2: enter();
            break;
          case 3: deleteSingleItem( &start, &last );
            break;
          case 4: list();
            break;
          case 5: find_name();
            break;
```

```
          case 6: find_store();
            break;
          case 7: find_date();
            break;
          case 8: write_list();
            break;
         case 9: save(); saveDataBackup();
            break;
         case 10: measures();
            break;
         case 11: modify();
            break;
         case 12: exit(0);
         case 13: save(); saveDataBackup(); exit(0);
        }//switch
       }//for

   return;
} //main


/* select an operation */
menu_select()  {
  char s[80];
  int c;
  clrscr();
  printf("\n              Grocery/Gas Shopping Tool  update: 20251015");
  printf("\n\n\n");
  printf("                1.  Create a shopping list\n                    creates file: \"yyyymmdd.shp\"\n");
  printf("                2.  Enter new items to the data list\n");
  printf("                3.  Delete a single item from data list\n");
  printf("                4.  Show current data list\n                    creates file: \"yyyymmdd.dat\"\n");
  printf("                5.  Search item by its name\n");
  printf("                6.  Search by grocery store\n");
  printf("                7.  Search by purchase date\n");
  printf("                8.  List and Count individual items\n                    creates file:
\"yyyymmdd.itm\"\n");
  printf("                9.  Save data list\n                creates file: \"grocdat3\"\n");
  printf("               10.  Measure conversion\n");
  printf("               11.  Remove and re-enter an item\n");
  printf("               12.  Exit without saving data list\n");
  printf("\n              13.  Save data list and Exit\n");
```

```
   do {
   printf("\n            Enter your choice:   ");
   gets(s);
   c = atoi(s);
  }//do
  while ( c < 0 || c > 13 );
  return c;
}//menu_select function


/* enter data for groceryItem's  */
void enter(void)  {
  int qualityRating = 0;
  int expCount = 0;
  int use_previous_expiration = 0;
  float actualPrice = 0;
  float costperUnit = 0;
  char buffer[30] = "";

  struct groceryItem *info;

  for( ; ; )  {
   info = ( struct groceryItem * )malloc(sizeof( list_entry ) );
   if(!info)  {
        printf("\n out of memory");
        return;
        }//if(!info)

      /* start inputing data to the node */
            clrscr();
            printf("\n\n");
            printf("\n    Enter the item name or type Quit to quit entry: ");
         scanf("%s", buffer);
         strcpy(info->item_Name, buffer);

      /* save a copy of the item name to search_name */
         strcpy(search_name, info->item_Name);

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

```
        printf("\n          Enter the item brand:   ");
      scanf("%s", buffer);
      strcpy(info->item_Brand, buffer);

        printf("\n          Enter name of the store:   ");
      scanf("%s", buffer);
      strcpy(info->item_storeName, buffer);

        /* for this element we will search the list for a previous purchased item and
         * use the expiration from that, so user doesn't have to always remember.
         * If nothing is found then we'll just enter it in by hand */

        search4_item_expiration();
    /* if a valid expiration date found then copy result to info->item_days_until_expiration */
    if (expires > 0 )  {  printf("              Found same item with expiration date. of %d days.\n
Do you want to use it? \(1 = yes 0 = no\)", expires);
                printf("\n              Enter 1 or 0: ");
                scanf( "  %d", &use_previous_expiration);
                 if(use_previous_expiration == 1)  { info->item_days_until_expiration = expires;
                   printf( "          Previous expiration used for this item. \n          press any key to continue.
\n",
                   info->item_days_until_expiration );  getch(); }
                    else { expires = 0; use_previous_expiration = 0; }
               }

    /* if nothing found as above then just manually enter the expiration days */
    if(expires == 0)  {
          printf("\n          Enter expiration days:   ");
          scanf("%d", &expCount);
     info->item_days_until_expiration = expCount;
      }

      printf("\n          Enter the cost per unit:   ");
      scanf("%f", &costperUnit );
    info->item_UnitCost = costperUnit;

      printf("\n          Enter the actual price paid per item:   ");
      scanf("%f", &actualPrice );
    info->item_actualprice_paid = actualPrice;

      printf("\n          Enter how unit cost calculated:\n");
```

```
        printf("            29 characters maximum:   ");
        scanf("%s", buffer);
        strcpy(info->item_Comments, buffer);

        printf("\n          Enter the date of purchase\n");
        printf("            the format for entry is yyyymmdd:   ");
        scanf("%s", buffer);
        strcpy(info->item_DatePurchased, buffer);

        printf("\n          Enter a quality rating from 1 to 5:   ");
           scanf("%d", &qualityRating);
      info->item_quality_rating = qualityRating;

        printf("\n          Enter the list, City Area:   ");
        scanf("%s", buffer);
        strcpy(info->item_City, buffer);

      Make_Linked_List(info, &start, &last);
    }//for
}//enter function


/* Create a doubly linked list in sorted order */
void Make_Linked_List( struct groceryItem *i,     //new element
   struct groceryItem **start,                //first element in the list
   struct groceryItem **last )                //last element in the list
   {
           struct groceryItem *old, *p;

           if(*last == NULL)  {              //first element in the list
             i->next = NULL;
             i->prior = NULL;
             *last = i;
             *start = i;
             return;
             }

  p = *start;                          //start at top of the list
  old = NULL;

  while(p)  {
   if(strcmp(p->item_Name, i->item_Name) < 0)  {
```

```
        old = p;
        p = p->next;
      }
      else  {
        if(p->prior)  {
          p->prior->next = i;
          i->next = p;
          i->prior = p->prior;
          p->prior = i;
          return;
        }
      i->next = p;                    //new first element
      i->prior = NULL;
      p->prior = i;
      *start = i;
      return;
      }  //end else

  }  //end while
  old->next = i;                    //put on end
  i->next = NULL;
  i->prior = old;
  *last = i;

} //end Make_Linked_List function


/* remove an element from the list */
void deleteSingleItem( struct groceryItem **start,  struct groceryItem **last )  {
      struct groceryItem *info, *find_single_item();
      char n[30];
      char d[9];
      char s[30];
      char buffer[30];

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

      printf("\n          Enter item date purchased:  ");
      scanf("%s", buffer);
      strcpy(d, buffer);
```

```
    printf("\n            Enter store item bought from:  ");
    scanf("%s", buffer);
    strcpy(s, buffer);

    info = find_single_item( n, d, s );

    if(info) {
        if(*start == info) {
            *start = info->next;
            if(*start)  (*start)->prior = NULL;
            else  *last = NULL;
            }
         else {
             info->prior->next = info->next;
             if(info != *last)  info->next->prior = info->prior;
             else  *last = info->prior;
             }
        printf("   DELETED - %s unit cost: %3.3f Purchased on: %s at %s\n", info->item_Name,
        info->item_UnitCost, info->item_DatePurchased, info->item_storeName);
        free(info);
        }
    getch();
    }  //end delete

/* find an item and make changes to the expiration date */
void modify(void) {
  int qualityRating = 0;
  int expCount = 0;
  int use_previous_expiration = 0;
  float actualPrice = 0;
  float costperUnit = 0;
  char buffer[30] = "";

  struct groceryItem *info;

   deleteSingleItem( &start, &last);

   info = ( struct groceryItem * )malloc(sizeof( list_entry ) );
   if(!info)  {
       printf("\n out of memory");
       return;
```

```
    }//if(!info)

  /* start inputing data to the node */
      clrscr();
      printf("\n\n");
      printf("\n      Re-enter the item name or type Quit to quit entry:   ");
    scanf("%s", buffer);
    strcpy(info->item_Name, buffer);

  /* save a copy of the item name to search_name */
    strcpy(search_name, info->item_Name);

  /* stop entry if Quit is typed in */
    if( strcmp(info->item_Name, "Quit" ) == 0) return;
      if( strcmp(info->item_Name, "quit" )     == 0) return;
      if( strcmp(info->item_Name, "QUIT" )  == 0) return;

      printf("\n          Enter the item brand:   ");
    scanf("%s", buffer);
    strcpy(info->item_Brand, buffer);

      printf("\n          Enter name of the store:   ");
    scanf("%s", buffer);
    strcpy(info->item_storeName, buffer);

      /* for this element we will search the list for a previous purchased item and
       * use the expiration from that, so user doesn't have to always remember.
       * If nothing is found then we'll just enter it in by hand */

      search4_item_expiration();
  /* if a valid expiration date found then copy result to info->item_days_until_expiration */
  if (expires > 0 )  {  printf("              Found same item with expiration date. of %d days.\n
Do you want to use it? \(1 = yes 0 = no\)", expires);
              printf("\n              Enter 1 or 0: ");
              scanf( "  %d", &use_previous_expiration);
               if(use_previous_expiration == 1)  { info->item_days_until_expiration = expires;
                 printf( "          Previous expiration used for this item. \n          press any key to continue.
\n",
                 info->item_days_until_expiration );  getch(); }
                  else { expires = 0; use_previous_expiration = 0; }
            }
```

```
    /* if nothing found as above then just manually enter the expiration days */
    if(expires == 0)  {
            printf("\n          Enter expiration days:   ");
            scanf("%d", &expCount);
     info->item_days_until_expiration = expCount;
       }

       printf("\n          Enter the cost per unit:   ");
       scanf("%f", &costperUnit );
    info->item_UnitCost = costperUnit;

       printf("\n          Enter the actual price paid per item:   ");
       scanf("%f", &actualPrice );
    info->item_actualprice_paid = actualPrice;

       printf("\n          Enter how unit cost calculated:\n");
       printf("             29 characters maximum:   ");
       scanf("%s", buffer);
       strcpy(info->item_Comments, buffer);

       printf("\n          Enter the date of purchase\n");
       printf("             the format for entry is yyyymmdd:   ");
       scanf("%s", buffer);
       strcpy(info->item_DatePurchased, buffer);

       printf("\n          Enter a quality rating from 1 to 5:   ");
          scanf("%d", &qualityRating);
    info->item_quality_rating = qualityRating;

       printf("\n          Enter the list, City Area:   ");
       scanf("%s", buffer);
       strcpy(info->item_City, buffer);

    Make_Linked_List(info, &start, &last);

}//enter function

/* find a single item to delete */
struct groceryItem *find_single_item( char *n, char *d, char *s ) {
    struct groceryItem *info;

   info = start;
```

```
      while( info ) {
            if( (strcmp( n, info->item_Name ) == 0 )  &&
              (strcmp( d, info->item_DatePurchased ) == 0 ) &&
               (strcmp( s, info->item_storeName ) == 0) )   { return info; }
            info = info->next;
       }
       if(!info) printf("              nothing found " );
       return NULL;
      }  //end findname


/* find a grocery Item by its name */
struct groceryItem *find_name() {
      struct groceryItem *info;
      int count = 0;
      int found = 0;
      int total_found  = 0;
      int display_count = 0;
      char name[30];
      char buffer[30];

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

      info = start;

      while( info ) {
            if(strcmp( name, info->item_Name ) == 0 )  { found = TRUE; printf("   %s unit cost: %3.2f
Purchased on: %s at %s\n",
                  info->item_Name, info->item_UnitCost, info->item_DatePurchased, info-
>item_storeName);  }
            //advance the counters when an item found
            if( found > 0) { total_found += 1; display_count += 1; }
            //displays set number of items before continuing and reset the display counter
            if(display_count == 3) { display_count = FALSE;  printf("\n"); getch(); }
            info = info->next;
          //reset the found flag to 0
          found = FALSE;
          count += 1;
       }
```

```
        /* print the number of items in list */
    if( total_found > 0 ) printf("\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


/* find a grocery Item by the store name */
struct groceryItem *find_store() {
      struct groceryItem *info;
      int count = 0;
      int found = 0;
      int total_found  = 0;
      int display_count = 0;
      char store[30];
      char buffer[30];

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

      info = start;

      while( info ) {
           if(strcmp( store, info->item_storeName ) == 0 )  { found = TRUE; printf("   %s unit cost: %3.2f
Purchased on: %s at %s\n",
                 info->item_Name, info->item_UnitCost, info->item_DatePurchased, info-
>item_storeName); }
           //advance the counters when an item found
           if( found > 0) { total_found += 1; display_count += 1; }
           //displays set number of items before continuing and reset the display counter
           if(display_count == 3) { display_count = FALSE;  printf("\n"); getch(); }
           info = info->next;
           //reset the found flag to 0
           found = FALSE;
           count += 1;
           }

      /* print the number of items in list */
```

```
   if( total_found > 0 ) printf("\n   Searched %d items for the store name and found %d matching",
count, total_found );
   printf("\n\n");
      if( total_found == 0 ) printf("              nothing found " );
      getch();
      return NULL;
      }  //end find_store


/* find a grocery Item by date of purchase */
struct groceryItem *find_date() {
      struct groceryItem *info;
      int count = 0;
      int found = 0;
      int total_found  = 0;
      int display_count = 0;
      char date[30];
      char buffer[30];

      printf("\n            Enter the purchase date of item:   ");
      scanf("%s", buffer);
      strcpy( date, buffer );

      info = start;

      while( info ) {
            if(strcmp( date, info->item_DatePurchased ) == 0 )  { found = TRUE; printf("   %s unit cost:
%3.2f Purchased on: %s at %s\n",
                  info->item_Name, info->item_UnitCost, info->item_DatePurchased, info-
>item_storeName); }
            //advance the counters when an item found
            if( found > 0) { total_found += 1; display_count += 1; }
            //displays set number of items before continuing and reset the display counter
            if(display_count == 3) { display_count = FALSE;  printf("\n"); getch(); }
            info = info->next;
            //reset the found flag to 0
            found = FALSE;
            count += 1;
            }

      /* print the number of items in list */
   if( total_found > 0 ) printf("\n   Searched %d items for the date of purchase entered and found %d
```

```
matching", count, total_found );
   printf("\n\n");
      if( total_found == 0 ) printf("              nothing found " );
      getch();
      return NULL;
      }  //end find_date

/* find and remove expired items by their expiration date */
struct groceryItem *remove_expired( struct groceryItem **start, struct groceryItem **last ) {
struct groceryItem *info;

int count = 0;
int found = 0;
int total_found = 0;
int display_count = 0;
char buffer[30];
int set_flag = 0;  //opens file only once if an item found

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

FILE *rem;

/* set the current date for calculating how long items have been in list */
/* ask user to input twice to verify the date if wrong then try again */
 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", buffer);
   if(strcmp(TodaysDate, buffer) != 0) {
     clrscr();
     printf("              Dates do not match - press any key to continue\n ");  getch(); }
     else break; }

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

 clrscr();
```

```
 info = *start;   // points to **start

 for( ; ; ) {
/* transform date string into (int) days (int) months (int) years */
   parce_and_convert_into_numbers(info->item_DatePurchased, TodaysDate);

/* see if leap year */
   is_it_leap_yr(the_Purchase_year);
   is_it_leap_yr(the_Current_year);

/* decide which function to use to calculate the number of days */
   if(the_Current_year == the_Purchase_year)calculate_same_yr_days();
   if(the_Current_year != the_Purchase_year)calculate_different_yr_days();

/* what to do if expired item is found */
   if(DaysCounted >= info->item_days_until_expiration) { found ++;

/* open the .rem file for writing data.  Changed so that file only gets created
 * when there are items to remove. */
   if(set_flag == 0) { set_flag= 1; rem = fopen( newfilename,"w"); }

/* print the item to delete first or else info->item_Name is corrupted */
   printf("    %s, stored in list for %d days \n    purchased on %s was set to expire in %d days is \n
now removed \n\n",
   info->item_Name, DaysCounted, info->item_DatePurchased, info->item_days_until_expiration );
   fprintf(rem,"    %s, stored in list for %d days \n    purchased on %s was set to expire in %d days is
\n    now removed \n\n",
   info->item_Name, DaysCounted, info->item_DatePurchased, info->item_days_until_expiration );

/* removes the item from the linked list */
   deleteItems( &start, &last, info );

/* advance the line display counters when an item found */
   if( found > 0) { total_found ++; display_count ++; }

/* displays set number of items before continuing and reset the display counter */
   if(display_count == 3) { display_count = FALSE; printf("\n"); getch(); }
   } // end if item found

/*reset the found flag to 0 */
   found = FALSE;
```

```
/* increment the loop count */
   count ++;
/* go to the next node */
   info = info->next;

  if( (info->next == NULL) && (total_found > 0) ) {
/* print out the final information */
   printf("          The text file: %s has been created \n", newfilename );
   fprintf(rem, "          %d expired items removed from list.  ", total_found ); getch();  fclose( rem );
break; }

  if( (info->next == NULL) && (total_found == 0) ) { printf("                    No items removed removed from
list.\n                    Press a key to continue\n", total_found ); getch(); break; }
  } //end for loop

} //end *remove_expired


/* remove all expired items from the list */
void deleteItems( struct groceryItem **start,  struct groceryItem **last, struct groceryItem *info )  {
  if( info ) {
             if( *start == info ) {
                 *start = info->next;
                 if( *start )  ( *start )->prior = NULL;
                  else  *last = NULL;
                 }
             else {
                   info->prior->next = info->next;
                   if( info != *last )  info->next->prior = info->prior;
                    else  *last = info->prior;
                   }

            free(info);
        }

}  //end deleteItems


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

```
    char buffer_previousName[30] ="empty";
    char buffer[30];
    char City[30];
    int count = 0;
  /* this enables programmer to adjust the expirations */
  float expiration_multiplier;

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

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

    /* set info pointer to beginning */
    info = start;
    /* get the city area from user */
    clrscr();
    printf("\n                Enter city area to list:  ");
    scanf("%s", buffer);
    strcpy(City, buffer);
    /* set the expiration multiplier for suggested item expiration date..prints a list at end of report */
    printf("\n                Set purchase frequency...");
    printf("\n       Usually a number between 2.5 and 3 (2.9 works good):  ");
    scanf("%f", &expiration_multiplier);

    printf("            Printing the list of items from ...\"grocdat3\"\n\n ");

    text_listing=fopen(newfilename,"w");

    fprintf(text_listing, "  Exp. Item                    Date printed:  %s\n\n", TodaysDate);
    while(info)  {
        if( strcmp( info->item_Name, "000000" ) != 0  && strcmp( info->item_Name, "zzzzzz" ) != 0
          && strcmp( info->item_City, City ) == 0) {
            if( strcmp( info->item_Name, buffer_previousName) != 0 )  {
        /* print the item and then a line for writing notes */
        fprintf(text_listing,"  %d  %s  \(%s\)\n\n\n", info->item_days_until_expiration, info->item_Name,
info->item_Comments  );

        /* increment count for each item found */
```

```
            /* get the next item */
            count +=1; } }

            strcpy(buffer_previousName, info->item_Name );


            info = info->next;
         }

    /* print the number of items in list */
    fprintf(text_listing, "\n\n          Expiration is %3.2f x the purchase frequency\n\n",
expiration_multiplier );
    fprintf(text_listing,"  Purchase made every 7 days.....set expiration to %3.0f days\n", (7 *
expiration_multiplier));
    fprintf(text_listing,"  Purchase made every 14 days.....set expiration to %3.0f days\n", (14 *
expiration_multiplier));
    fprintf(text_listing,"  Purchase made every 21 days.....set expiration to %3.0f days\n", (21 *
expiration_multiplier));
    fprintf(text_listing,"  Purchase made every 28 days.....set expiration to %3.0f days\n", (28 *
expiration_multiplier));
    fprintf(text_listing,"  Purchase made every 35 days.....set expiration to %3.0f days\n", (35 *
expiration_multiplier));
    fprintf(text_listing,"  Purchase made every 42 days.....set expiration to %3.0f days\n", (42 *
expiration_multiplier));
    fprintf(text_listing,"  Purchase made every 63+ days.....set expiration to %3.0f days\n\n", (65 *
expiration_multiplier));
    fprintf(text_listing,"  Item quality 1 - poor 2 - fair 3 - average 4 - good 5 - very good\n\n");
    fprintf(text_listing,"  There are currently %d different items in the grocery list database", count);
    fprintf(text_listing,"\n\n");
        /* make room to add some new items not in list */
    fprintf(text_listing,"  New items to add..");

       fclose(text_listing);

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

/* find the best price for items that are in array
* that is created as a grocery shopping list */
struct groceryItem *findbest() {
```

```
    struct groceryItem *info;

/* a place to store the cheapest item */
    struct storageItem {
    char name[30] ;
    char brand[30];
    char store[30];
    char price_units_compare[30];
    char date_of_purchase[9];
    float price;
    float totalpaid;
    };

 /* grouping of items in the shopping list:  name is manually input, but
     * cheapest, and store are retrieved from the LL and saved when found */
    struct shopping_item {
    char name[30] ;
    };

    /* declare array to store shopping list of up to 26 items and best item */
    /* the first array is to hold the entered shopping list */
    struct shopping_item shop[25] = {""};
 /* the next 2 items are of type "storageItem" and save the
  * cheapest, and most expensive prices */
struct storageItem best[1];
struct storageItem expensive[1];

char buffer[30];
int Match = 1;
char City[30];
int matching_city = -1;

int PrintedFirst = 0;
int searchFlag = 0;

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

    /* make TRUE and FALSE constants */
    int YES = 1;
```

```
    int NO = 0;

    int STOP = NO;
    int index = 0;   //index to move through the shopping items
    int count = 0;  //counts times the index was moved


  /* these three variables are used to print the grocery shopping list
   * either all matching or just the cheapest ones are printed - NOT BOTH */
  int print_all;
    int print_cheapest;
  int choice;

    /* variable to hold total of best price and amount saved on an individual item */
    float total_shopping_price = 0;
    float amount_saved = 0;
    /* the actual size of package, can, jar, etc. */
    float unitsize = 0;
    /* make a unique file name for the file  */
  strcpy(newfilename, TodaysDate);
  strncat(newfilename, file_extension, 4);

  clrscr();

    printf("\n                Enter city area to shop:  ");
    scanf("%s", buffer);
    strcpy(City, buffer);

    do {
      /* enter the items and type quit to quit entry */
      printf("\n          Enter item name for %d of number of 24:  ", (index + 1) );  //index actually starts
with zero

    /* input the new value for buffer */
    scanf("%s",  buffer);
      strcpy( shop[index].name,  buffer);

      if(strcmp(shop[index].name,  "Quit" ) == 0)  { STOP = YES; clrscr();
         printf("                Quitting entry .. moving to search \n" );  getch(); }
        else if(strcmp(shop[index].name, "QUIT") == 0)  { STOP = YES; clrscr();
         printf("                Quitting entry .. moving to search \n" );  getch(); }
        else if(strcmp(shop[index].name, "quit") == 0)  { STOP = YES; clrscr();
```

```
          printf("                Quitting entry .. moving to search \n" );  getch(); }
        else { index += 1;   count = index; }
        if( (count + 1) == 25)  { STOP =YES; clrscr();
          printf("                Max entries reached .. moving to search \n" );  getch(); }
    } while ( STOP == NO ); //end do-while


  printf("\n              You have a shopping list size of %d items.\n", count );
  /* reset the STOP command */
  STOP = NO;

  /* enter the way you would like to print the searched shopping items */
  printf("\n              Enter 0: To print all matching items");
    printf("\n              Enter 1: To print best priced items:   ");
    scanf("%d", &choice);
     if(choice == 0) { print_all =YES; print_cheapest = NO; }
     if(choice == 1) { print_all = NO; print_cheapest = YES; }

    /* open text file for write - clears old file */
  shp = fopen(newfilename,"w");

  /* set the info pointer to beginning of list and clear the screen */
  info = start;

    clrscr();

  fprintf(shp, "            Shopping list for          >> %s  <<\n\n", TodaysDate);

     for(index = 0; index < count; ) {

          do {
                Match = strcmp(shop[index].name, info->item_Name);
                matching_city = strcmp( info->item_City, City );
                if( info->next == NULL ) { info = start;  searchFlag += 1; }
              if(print_cheapest == YES) {
              if( ( searchFlag  > 0 ) && ( Match !=0 ) ) { index += 1; searchFlag = 0; info = start;
break; } }
              if(print_all == YES) {
              if( ( searchFlag  > 0 ) && ( Match !=0 ) ) { index += 1; searchFlag = 0; info = start;
break; } }
               /* Next line needs the the "if" statement so it doesn't advance the pointer on exit */
              if( Match != 0 ) info = info->next;
```

```
                } while (Match != 0);

            if( print_all == YES ) {
                if( PrintedFirst == 0 )  { fprintf( shp, "  First item: %s\n\n", shop[0].name);
               /*printf("  First item: %s\n\n", shop[0].name);*/ PrintedFirst = 1; }

             /* matching item found */
             if( ( Match == 0 ) && (matching_city == 0 ) ) {

               /* print the item found to the file and screen */
               fprintf( shp, "  %s %s, %3.3f %s from %s on %s\n\n",
                  info->item_Brand, info->item_Name, info->item_UnitCost, info->item_Comments,
                   info->item_storeName, info->item_DatePurchased );



              } // if Match == 0
          } // if print_all == TRUE


        if( print_cheapest == YES ) {
                 /* this statement only gets printed once and we initialize best[0].price to a high price
once
                  * this is done so that the first price gets copied */
                 if( PrintedFirst == 0 )  { best[0].price = 50.00; expensive[0].price = 0.00;
                            fprintf( shp, "  First item: %s\n\n", shop[0].name);
                /*printf("  First item: %s\n\n", shop[0].name);*/ PrintedFirst = 1; }

              /* matching item found */
               if( ( Match == 0 ) && (matching_city == 0 ) ) {

                 /* save the cheaper item */
                 if( info->item_UnitCost != best[0].price ) {
                   if( info->item_UnitCost < best[0].price ) {
                 best[0].price = info->item_UnitCost;
                 best[0].totalpaid = info->item_actualprice_paid;
                 strcpy(best[0].name, info->item_Name );
                 strcpy(best[0].brand, info->item_Brand );
                 strcpy(best[0].store, info->item_storeName );
                 strcpy(best[0].date_of_purchase, info->item_DatePurchased );
                 strcpy(best[0].price_units_compare, info->item_Comments );
                 } }
```

```
                /* save the more expensive item */
                if( info->item_UnitCost != expensive[0].price); {
                        if( info->item_UnitCost > expensive[0].price) {
                        expensive[0].price = info->item_UnitCost;
                        expensive[0].totalpaid = info->item_actualprice_paid;
                        strcpy(expensive[0].name, info->item_Name );
                strcpy(expensive[0].brand, info->item_Brand );
                strcpy(expensive[0].store, info->item_storeName );
                strcpy(expensive[0].date_of_purchase, info->item_DatePurchased );
                      strcpy(expensive[0].price_units_compare, info->item_Comments );
                      } }

              } // if Match == 0
            } // if print_cheapest == YES

              /* the remaining part of this function shifts to the next item in shopping list if
               * certain conditions are met */
               if( ( Match != 0 ) && ( index <= count ) ) {

                        /* determine if the next item is "quit" and stop the printing */
                        if(strcmp(shop[index].name,  "Quit" ) == 0) { STOP = YES; }
               if(strcmp(shop[index].name, "QUIT") == 0) { STOP = YES; }
               if(strcmp(shop[index].name, "quit") == 0) { STOP = YES; }

              /* but first we must print the previous item and its info if found */
                        if ( (print_cheapest == YES ) && ( best[0].price != 50.00 ) || ( expensive[0].price !=
0 ) ) {

                      /* save the running total cost of the groceries found and the individual amount that
was saved
                       * when comparing an item with another */
                      total_shopping_price += best[0].totalpaid;
                      amount_saved = (expensive[0].totalpaid - best[0].totalpaid);

                     /* print the next items found to the .shp file */
                if ( (strcmp(expensive[0].store, best[0].store) != 0 ) && (amount_saved >= 0) )
{  fprintf( shp, "  %s %s, %3.3f %s, units: %3.0f from %s on %s paid $%3.2f \n  %s %s at %3.3f %s,
units: %3.0f from %s on %s was\n  the most expensive item.  You save $%3.2f at %s.\n\n ",
                        best[0].brand, best[0].name, best[0].price, best[0].price_units_compare,
(best[0].totalpaid/best[0].price), best[0].store, best[0].date_of_purchase, best[0].totalpaid,
expensive[0].brand, expensive[0].name,
                        expensive[0].price, expensive[0].price_units_compare, (expensive[0].totalpaid/
```

```
expensive[0].price), expensive[0].store, expensive[0].date_of_purchase, amount_saved,
best[0].store );   }

                    if ( (strcmp(expensive[0].store,best[0].store) != 0 ) && (amount_saved < 0) )
{  fprintf( shp, "  %s %s, %3.3f %s, units: %3.0f from %s on %s paid $%3.2f \n  %s %s at %3.3f %s,
units: %3.0f from %s on %s was\n  the most expensive item.  Paid a cheaper price at %s because\n  of
packaging but cheapest price %s is at %s.\n\n ",
                      best[0].brand, best[0].name, best[0].price, best[0].price_units_compare,
(best[0].totalpaid/best[0].price), best[0].store, best[0].date_of_purchase, best[0].totalpaid,
expensive[0].brand, expensive[0].name,
                      expensive[0].price, expensive[0].price_units_compare, (expensive[0].totalpaid/
expensive[0].price), expensive[0].store, expensive[0].date_of_purchase, expensive[0].store,
best[0].price_units_compare, best[0].store ); }

                    if ( (strcmp(expensive[0].store, best[0].store) == 0 ) && (amount_saved > 0) )
{  fprintf( shp, "  %s %s, %3.3f %s, units: %3.0f from %s on %s paid $%3.2f \n  %s %s at %3.3f %s,
units: %3.0f from %s on %s was\n  the most expensive item. Buy the %s brand for best price.\n\n ",
                      best[0].brand, best[0].name, best[0].price, best[0].price_units_compare,
(best[0].totalpaid/best[0].price), best[0].store, best[0].date_of_purchase, best[0].totalpaid,
expensive[0].brand, expensive[0].name,
                      expensive[0].price, expensive[0].price_units_compare, (expensive[0].totalpaid/
expensive[0].price), expensive[0].store, expensive[0].date_of_purchase, best[0].brand);   }

                    if ( (strcmp(expensive[0].store, best[0].store) != 0 ) && (amount_saved == 0) &&
strcmp(expensive[0].date_of_purchase,best[0].date_of_purchase) != 0 ) {  fprintf( shp, "  %s %s, %3.3f
%s, units: %3.0f from %s on %s paid $%3.2f \n  %s %s at %3.3f %s, units: %3.0f from %s on %s was\n
the most expensive item. The dollar amounts are the same.\n\n ",
                      best[0].brand, best[0].name, best[0].price, best[0].price_units_compare,
(best[0].totalpaid/best[0].price), best[0].store, best[0].date_of_purchase, best[0].totalpaid,
expensive[0].brand, expensive[0].name,
                      expensive[0].price, expensive[0].price_units_compare, (expensive[0].totalpaid/
expensive[0].price), expensive[0].store, expensive[0].date_of_purchase);   }

                    if ( (strcmp(expensive[0].store, best[0].store) == 0 ) &&
strcmp(expensive[0].date_of_purchase,best[0].date_of_purchase) == 0)  {  fprintf( shp, "  %s %s, %3.3f
%s, units: %3.0f from %s on %s paid $%3.2f\n  This item doesn't have any other store to compare with.
\n\n ",
                      best[0].brand, best[0].name, best[0].price, best[0].price_units_compare,
(best[0].totalpaid/best[0].price), best[0].store, best[0].date_of_purchase, best[0].totalpaid);    }

                    if ( (strcmp(expensive[0].store, best[0].store) == 0 ) && (amount_saved < 0) &&
strcmp(expensive[0].store,
```

```
                    best[0].store) != 0 )  {  fprintf( shp, "  %s %s, %3.3f %s, units: %3.0f from %s on %s
paid $%3.2f \n  %s %s at %3.3f %s, units: %3.0f from %s on %s was\n  the most expensive item.\n\n
",
                     best[0].brand, best[0].name, best[0].price, best[0].price_units_compare,
(best[0].totalpaid/best[0].price), best[0].store, best[0].date_of_purchase, best[0].totalpaid,
expensive[0].brand, expensive[0].name,
                     expensive[0].price, expensive[0].price_units_compare, (expensive[0].totalpaid/
expensive[0].price), expensive[0].store, expensive[0].date_of_purchase);   }

                  }  //end..if ( (print_cheapest == YES ) && ( best[0].price != 50.00 ) ||
( expensive[0].price != 0 ) )

                     /* we cant find any more of a shopping item so move to the next item */
                     if( STOP != YES )  {
                       if ( Match == 0) { fprintf( shp," Next/New item: %s\n\n", shop[index].name);  }
                      else {  fprintf( shp,"  Next/New item: %s\n\n", shop[index].name);  }

                     /* reset both of the storage arrays for highest and lowest price */
                     strcpy(best[0].brand, NULL); strcpy(best[0].name, NULL);
                     strcpy (best[0].store, NULL); strcpy(best[0].date_of_purchase, NULL);
                  strcpy(best[0].price_units_compare, NULL);
                  best[0].price = 50.00;
                  strcpy(expensive[0].brand, NULL); strcpy(expensive[0].name, NULL);
                        strcpy (expensive[0].store, NULL); strcpy(expensive[0].date_of_purchase, NULL);
                  strcpy(expensive[0].price_units_compare, NULL);
                  expensive[0].price = 0.00;
                 }
                } //if match !=0

                  /* this resets the searchFlag, allowing progression through the LL at the beginning,
move to next item
                  * in LL and resets the "Match" variable allowing searching for a "Match==0" condition,
which is a match */
               if (print_all == YES ) { searchFlag = 0; }

               info = info->next;
                  Match = -1;
                    matching_city = -1;


               /* if at the end of the shopping list then break from for loop */
               if ( index == count )  {
```

```
                    break; }
                } //end for loop

        /* clean up some items */
        /* this prevents an item being printed as lowest/highest when ALL matching items
         * are being printed.  This happens when you print lowest/highest, then select print ALL
         * matching items. */
        /* reset both of the storage arrays for highest and lowest price */
                        strcpy(best[0].brand, NULL); strcpy(best[0].name, NULL);
                        strcpy (best[0].store, NULL); strcpy(best[0].date_of_purchase, NULL);
                    strcpy(best[0].price_units_compare, NULL);
                    best[0].price = 50.00;
                    strcpy(expensive[0].brand, NULL); strcpy(expensive[0].name, NULL);
                        strcpy (expensive[0].store, NULL); strcpy(expensive[0].date_of_purchase, NULL);
                    strcpy(expensive[0].price_units_compare, NULL);
                    expensive[0].price = 0.00;

        /* print summary */
        if(print_cheapest == YES) {
        fprintf( shp, "\n         Your estimated cost for groceries today is $%3.2f ", total_shopping_price );
    /*  printf("\n         Your estimated cost for groceries today is $%3.2f ", total_shopping_price ); */  }

        fprintf( shp, "\n         List complete. You have selected %d items to shop for. ", count );
      printf("\n       Done building list. Your shopping list saved to %s ", newfilename );
      fclose( shp );
      getch();
      return NULL;
    }  //end findbest

measures() {
/* start the menu */
  for( ; ; )  {
        switch(submenu1())  {
          case 1:  quarts_ounces_quarts();
            break;
          case 2:  quarts_ML_quarts();
            break;
          case 3:  ounces_ML_ounces();
            break;
          case 4:  ounces_Gal_ounces();
            break;
          case 5:  Gal_ML_Gal();
```

```
            break;
           case 6:  Gal_liters_Gal();
            break;
           case 7:  Pnts_ounces_Pnts();
            break;
           case 8:  return;
         }//switch
        }//for

    return;
} // end measures()

/* select an operation */
submenu1( ) {
  char s[80];
  int c;
  clrscr();
  printf("\n        Select a measurment to convert");
  printf("\n\n\n\n");
  printf("                  1.  Quarts->Ounces->Quarts\n");
  printf("                  2.  Quarts->ML->Quarts\n");
  printf("                  3.  Ounces->ML->Ounces\n");
  printf("                  4.  Ounces->Gallons->Ounces\n");
  printf("                  5.  Gallons->ML->Gallons\n");
  printf("                  6.  Gallons->Liters->Gallons\n");
  printf("                  7.  Pints->Ounces->Pints\n");
  printf("                  8.  Exit\n");
    do  {
    printf("\n             Enter your choice:   ");
    gets(s);
    c = atoi(s);
   }//do
  while ( c < 0 || c > 8 );
  return c;
}//submenu1 function

/* conversion functions */
quarts_ounces_quarts() {
       int select;
       float o, qt;
       clrscr();
       printf("          Select 1 for converting ounces. or 0 for converting quarts    ");
```

```
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                Enter the quarts to convert:    ");
            scanf(" %f", &qt); o = qt * 32; printf("\n                %3.2f quarts = %3.2f ounces", qt, o); }
    if( select == 1 ) {printf("\n                Enter the ounces to convert:   ");
            scanf(" %f", &o);  qt = o * .03125; printf("\n                %3.2f ounces = %3.2f quarts", o,
qt); }
    getch();
 }

quarts_ML_quarts() {
      int select;
      float ml, qt;
      clrscr();
      printf("      Select 1 for converting milliliters. or 0 for converting quarts    ");
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                Enter the quarts to convert:    ");
            scanf(" %f", &qt); ml = qt * 946.353; printf("\n                %3.2f quarts = %3.2f ml", qt,
ml); }
    if( select == 1 ) {printf("\n                Enter the milliliters to convert:   ");
            scanf(" %f", &ml); qt = ml * .001056688; printf("\n                %3.2f ml = %3.2f quarts",
ml, qt); }
    getch();
}

ounces_ML_ounces() {
   int select;
      float ml, o;
      clrscr();
      printf("      Select 1 for converting milliliters. or 0 for converting ounces    ");
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                Enter the ounces to convert:    ");
            scanf(" %f", &o); ml = o * 28.413; printf("\n                %3.2f ounces = %3.2f ml", o, ml); }
    if( select == 1 ) {printf("\n                Enter the milliliters to convert:   ");
            scanf(" %f", &ml); o = ml * .035195; printf("\n                %3.2f ml = %3.2f ounces", ml,
o); }
    getch();
}

ounces_Gal_ounces() {
      int select;
      float gal, o;
      clrscr();
```

```
      printf("      Select 1 for converting gallons. or 0 for converting ounces    ");
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                Enter the ounces to convert:    ");
            scanf(" %f", &o); gal = o * .0078125; printf("\n                %3.2f ounces = %3.2f gal.", o,
gal); }
    if( select == 1 ) {printf("\n                Enter the gallons to convert:   ");
            scanf(" %f", &gal); o = gal * 128.0; printf("\n                %3.2f gal. = %3.2f ounces", gal,
o); }
    getch();
}

Gal_ML_Gal() {
      int select;
      float gal,ml;
      clrscr();
      printf("      Select 1 for converting gallons. or 0 for converting milliliters    ");
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                Enter the milliliters to convert:    ");
            scanf(" %f", &ml); gal = ml * .000264172; printf("\n                %3.2f milliliters = %3.2f
gal.", ml, gal); }
    if( select == 1 ) {printf("\n                Enter the gallons to convert:   ");
            scanf(" %f", &gal); ml = gal * 3785.412; printf("\n                %3.2f gal. = %3.2f milliliters",
gal, ml); }
    getch();
}

Gal_liters_Gal() {
   int select;
      float gal,lt;
      clrscr();
      printf("      Select 1 for converting gallons. or 0 for converting liters    ");
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                Enter the liters to convert:    ");
            scanf(" %f", &lt); gal = lt * .2199693; printf("\n                %3.2f liters = %3.2f gal.", lt,
gal); }
    if( select == 1 ) {printf("\n                Enter the gallons to convert:   ");
            scanf(" %f", &gal); lt = gal * 4.54609; printf("\n                %3.2f gal. = %3.2f liters", gal,
lt); }
    getch();
}

Pnts_ounces_Pnts() {
```

```
   int select;
      float pnts, o;
      clrscr();
      printf("       Select 1 for converting pints. or 0 for converting ounces    ");
      scanf("   %d", &select);
       if( select == 0 ) { printf("\n                  Enter the ounces to convert:    ");
            scanf(" %f", &o); pnts = o * .0625; printf("\n                    %3.2f ounces = %3.2f pints", o,
pnts); }
    if( select == 1 ) {printf("\n                  Enter the pints to convert:   ");
            scanf(" %f", &pnts); o = pnts * 16.0; printf("\n                     %3.2f pints = %3.2f ounces",
pnts, o); }
    getch();
}

/* end of conversion functions */

/* look for expiration of a matching item in the list */
void search4_item_expiration(void) {
      struct groceryItem *info;
      char buffer_previousName[30] ="empty";
      char buffer[30];
       info = start;
    expires = 0;  //reset value to zero
/* search through the whole list for the item being inserted. */
/* Quit when the "next" pointer is NULL */
       while(info)  {
           if( strcmp( info->item_Name, "000000" ) != 0  && strcmp( info->item_Name, "zzzzzz" ) != 0 )  {
             if( strcmp( info->item_Name, buffer_previousName) != 0 )  {

               /* save the expiration date of this item */
               if( strcmp( info->item_Name, search_name) == 0 )  {
                 expires = info->item_days_until_expiration;  }
             }
         } // end outermost if statement
           strcpy(buffer_previousName, info->item_Name );

        info = info->next;
      }  //end while
}  //end  search4_item_expiration


/* display the entire list */
```

```
void list(void) {
      struct groceryItem *info;
      int count = 0;
      int display_count = 0;

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

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

   makelist = fopen(newfilename,"w");

      info = start;
      clrscr();
      fprintf(makelist, "   This is the entire listing of items in the file:  \"grocdat3 \"\n   Date: %s \n\n",
TodaysDate );
      printf("    This is the entire listing of items in the file: \"grocdat3\"");
      printf("\n\n\n\n");

      while(info)  {
            // the if statement prevents the dummy nodes from being printed
            if( strcmp( info->item_Name, "000000" ) != 0  && strcmp( info->item_Name, "zzzzzz" ) != 0  ) {
              printf("  %s  %3.3f  %s  %s  %s  %s  paid $%3.2f\n", info->item_Name,
                info->item_UnitCost, info->item_Comments, info->item_Brand, info->item_DatePurchased,
info->item_storeName, info->item_actualprice_paid );
               /* save this list to a file */
               fprintf(makelist,"  %s  %3.3f %s  %s  %s  %s paid $%3.2f\n", info->item_Name,
                info->item_UnitCost, info->item_Comments, info->item_Brand, info->item_DatePurchased,
info->item_storeName, info->item_actualprice_paid  );
            }
            /* 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 = FALSE;  printf("\n"); getch(); }
            info = info->next;
        }
   /* print the number of items in list */
```

```
   fprintf(makelist, "\n            There are currently %d items in the grocery list database", count);
   printf("\n  There are currently %d items in the grocery list database. \n  File: %s created.", count,
newfilename);
   printf("\n\n");
      fclose( makelist );
      if(count == 0) printf("                nothing found " );
      getch();
      }  //end list


/* save list to disk file */
void save( void ) {
      struct groceryItem *info;
      FILE *fp;

      fp = fopen( "grocdat3",  "wb" );
      if( !fp ) {
            printf( "cannot open file\n" );
            exit( 1 );
            }

      info = start;
      while( info ) {
            fwrite( info, sizeof( struct groceryItem ), 1, fp );
            info = info->next;
            }
      fclose( fp );
      printf( "\n         saved grocdat3..." );

      }     // end save


/* load file to list */
 void load( void )  {
       struct groceryItem *info;
       FILE *fp;

       fp = fopen( "grocdat3", "rb" );
       if( !fp ) {
            printf( "cannot open file\n" );
            exit( 1 );
             }
```

```
        /* free any previously allocated memory */
        while( start )  {
              info = start->next;
              free( info );
              start = info;
              }
        /* reset top and bottom pointers */
        start = last = NULL;

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

            Make_Linked_List( info, &start, &last );
              } //end while
        fclose( fp );
     printf( "              loaded file - press a key to continue" );
     getch();

    } // end load

void saveDataBackup( void ) {
    struct groceryItem *info;
        FILE *fp;

        fp = fopen( "grocdat3.bak",  "wb" );
        if( !fp ) {
              printf( "cannot open file\n" );
              exit( 1 );
              }

        info = start;
        while( info ) {
              fwrite( info, sizeof( struct groceryItem ), 1, fp );
              info = info->next;
              }
        fclose( fp );
        printf( "\n          saved grocdat3.bak - press a key to continue" );
```

```
    getch();

} // end saveDataBackup
```