```c
/* This is a grocery shopping program that uses a doubly linked list with menu
 * for adding, deleting, saving, items and reading items from a file. Expired items are
 * automatically purged from the list at launch but manually removing individual
 * items is also an available option. My written log has details of the new changes.
 * Programmed by Ron Lewis (Latest update: 20261109)
 */



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

/* New 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, perItem, perBottle, etc. )
    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
    int item_age;       // this is the number of days item has been in list  ADDED NEW 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);

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 "shopdata" file exists load it, otherwise skip to menu */
   fp = fopen( "shopdata", "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  New version update: 20261109");
```

```
  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: \"shopdata\"\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 or Exit 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;
               if( strcmp(info->item_Name, "Exit" ) == 0) break;
              if( strcmp(info->item_Name, "exit" )    == 0) break;
              if( strcmp(info->item_Name, "EXIT" )  == 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 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.\n");
            printf("            Where user controls the price paid, use $2:     ");
            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.\n");
            printf("            Where user controls the price paid, use $2:     ");
            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 from %s
on: %s\n",
                        info->item_Name, info->item_UnitCost, info->item_storeName, info->item_DatePurchased );  }
                //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 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

  /* print the start of the printout that isn't looped */
  /* don't open and overwrite the file if already created today */
  if( (rem = fopen(newfilename,"rb")) != NULL)  {
          printf("               the file %s was already created for today, \n                skipping operation",
newfilename);
          printf(" to prevent file overwriting.");
          fclose(rem); getch(); return; }

  /* open the .rem file for writing items removed data. After creation no file with the same name can be created
   * to prevent overwriting */
    if(set_flag == 0) { set_flag= 1; rem = fopen( newfilename,"w");
      fprintf(rem, "                  Item\(s\) that were removed today, %s\n\n", TodaysDate); }


/* loop through the items and remove the expire ones */
  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();

/* this is the new item being loaded with data from the DaysCounted calculation */
/* It gives the age of the item in the list. Must capture this number before the delete process */
    info->item_age = DaysCounted;

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

/* print the item to delete first or else info->item_Name is corrupted */
    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 );

/* increment the loop count */
    count ++;

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

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

/* go to the next node */
    info = info->next;

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

  if( (info->next == NULL) && (count == 0) ) { printf("                          No items removed removed from list.
\n                              Press a key to continue\n" ); 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 ...\"shopdata\"\n\n ");

        text_listing=fopen(newfilename,"w");

        fprintf(text_listing, "  Individual items listing.  Area:  %s\n", City );
        fprintf(text_listing, "  Date printed:  %s\n", TodaysDate );
        fprintf(text_listing, "  ITEM, EXPIRES, MEASURE  Note: EXPIRES may be different for same items.\n\n" );
        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 items and use an extra \t when string item name is less than 12 characters */
                if (strlen( info->item_Name ) < 12) {
                fprintf(text_listing,"    %s\t\t%d\t\t%s\n",
                  info->item_Name, info->item_days_until_expiration, info->item_Comments ); }
```

```
                if (strlen( info->item_Name ) >=12) {
                fprintf(text_listing,"    %s\t%d\t\t%s\n",
                  info->item_Name, info->item_days_until_expiration, 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 %3.1f 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;
        int age;
        };

    /* 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 24 items */
        struct shopping_item shop[24] = {""};

    /* the next 3 items are of type "storageItem" and save the
     * cheapest, most expensive, and most recent purchase */
    struct storageItem best[1];
```

```
    struct storageItem expensive[1];
    struct storageItem recent[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

    int print_flag = NO; // set to YES if any of the print statements are printed for cheapest/expensive item
        int print_force = NO; // is set when a NEW item is found, then resets after NEW statement printed

        /* variable to hold total of best price and amount saved on an individual item */
        float total_shopping_price = 0;
        float amount_saved = 0;
        float negative_amount_saved = 0;
        /* the actual size of package, can, jar, etc. */
        float unitsize_best = 0;
        float unitsize_expensive = 0;
        /* variables to do the calculation of the real_savings */
        float units_difference = 0;
```

```
        float price1 = 0;
        float price2 = 0;
        float evenUnits_savingsAmount = 0;
        float extra_cost_required = 0;
        float real_savings = 0;
        float savings_index = 0;

        /* variable to hold the tag for when a same store, price increase has been detected */
    int price_increase_detect = 0;

    strcpy(newfilename, TodaysDate);
    strncat(newfilename, file_extension, 4);

    clrscr();
        printf("\n              Enter \"quit\" as an item to terminate data entry.");
        printf("\n                      First, 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 23:  ", (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) == 23)  { STOP =YES; clrscr();
```

```
                printf("                        Max of %d entries reached .. moving to search \n", count );  getch(); }
        } while ( STOP == NO ); //end do-while


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

    printf("\n                       press any key to print shopping list"); getch();

        /* 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; ) {
                    /* reset the real savings variables */
                              units_difference = 0;
                        price1 = 0; price2 = 0;
                        evenUnits_savingsAmount = 0;
                        extra_cost_required = 0;
                        real_savings = 0;
                            savings_index = 0;


                    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( ( 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);

            /* this statement only gets printed once and we initialize best[0].price to a high price once
             * and expensive price to 0. This is done so that the first price gets copied */
             if( PrintedFirst == 0 )  { best[0].price = 100.00; expensive[0].price = 0.00; recent[0].age = 500;
                                   fprintf( shp, "  First item: %s\n\n", shop[0].name);
                  PrintedFirst = 1; }

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

                /* save the most recently purchased item */
                if( info->item_age < recent[0].age) {
                            recent[0].price = info->item_UnitCost;
                            recent[0].totalpaid = info->item_actualprice_paid;
                            recent[0].age = info->item_age;
                            strcpy(recent[0].name, info->item_Name );
                strcpy(recent[0].brand, info->item_Brand );
                strcpy(recent[0].store, info->item_storeName );
                strcpy(recent[0].date_of_purchase, info->item_DatePurchased );
                        strcpy(recent[0].price_units_compare, info->item_Comments );
                        }

                /* save the cheaper item */
                if( info->item_UnitCost < best[0].price ) {
                best[0].price = info->item_UnitCost;
                best[0].totalpaid = info->item_actualprice_paid;
                best[0].age = info->item_age;
                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) {
                                    expensive[0].price = info->item_UnitCost;
                                    expensive[0].totalpaid = info->item_actualprice_paid;
                                    expensive[0].age = info->item_age;
                                    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

                    /* 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; }


                                    if ( ( 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 */
                                   unitsize_best = best[0].totalpaid / best[0].price;
                                   unitsize_expensive = expensive[0].totalpaid / expensive[0].price;
                                   total_shopping_price += best[0].totalpaid;
                                   amount_saved = (expensive[0].totalpaid - best[0].totalpaid);
```

```
                        /* take amount saved and make a positive number called negative_amount_saved for printing
*/
                        if (amount_saved < 0) negative_amount_saved = amount_saved-(amount_saved*2);

                        /* calculations to find real_savings when larger unit size is required to get cheaper
unit price */
                        if( ( best[0].price < expensive[0].price) && (amount_saved < 0) ) {

                        /* calculate the difference in the unit sizes */
                            if ( unitsize_best > unitsize_expensive ) units_difference = unitsize_best -
unitsize_expensive;
                            if ( unitsize_best < unitsize_expensive ) units_difference = unitsize_expensive -
unitsize_best;


                          /* difference in units, div. by the more expensive item size...must be less than 1 */
                          price1 = unitsize_expensive * best[0].price;
                              price2 = unitsize_expensive * expensive[0].price;

                              /* the savings amount when you compare unit price with cheaper container size */
                              evenUnits_savingsAmount = price2 - price1;

                              /* the unit for unit savings divided by the extra cost required ... gives
percentage */
                              extra_cost_required = units_difference * best[0].price;
                              real_savings = evenUnits_savingsAmount / extra_cost_required;



                          /* look up the real_savings_index by sorting through pre index data */
                          if ( real_savings < .10 ) savings_index = 0;
                          if ( (real_savings >.10) && (real_savings < .20) ) savings_index = 1;
                          if ( (real_savings >.20) && (real_savings < .30) ) savings_index = 2;
                          if ( (real_savings >.30) && (real_savings < .40) ) savings_index = 3;
                          if ( (real_savings >.40) && (real_savings < .50) ) savings_index = 4;
                          if ( (real_savings >.50) && (real_savings < .60) ) savings_index = 5;
```

```
                                    if ( (real_savings >.60) && (real_savings < .70) ) savings_index = 6;
                                    if ( (real_savings >.70) && (real_savings < .80) ) savings_index = 7;
                                    if ( (real_savings >.80) && (real_savings < .90) ) savings_index = 8;
                                    if ( (real_savings >.90) && (real_savings < 1.0) ) savings_index = 9;
                                    if (real_savings >= 1.0) savings_index = 10;
                                   }

                                 /* print the next items found to the .shp file */
                              /* set the print_flag to YES in ANY of these statements get printed */

                               if ( (best[0].price != expensive[0].price)  && (strcmp(best[0].brand, expensive[0].brand) == 0)
&& (best[0].price != 100) && (expensive[0].price != 0) && (strcmp(best[0].store, expensive[0].store) == 0) &&
                                  ( best[0].age > expensive[0].age ) && ( units_difference < 1 ) && (amount_saved > 0)
                                   ||  (best[0].price != recent[0].price)  && (strcmp(best[0].brand, recent[0].brand) == 0) &&
(strcmp(best[0].store, recent[0].store) == 0) && (best[0].price != 100) && (expensive[0].price != 0) &&
                                  ( ( best[0].age > recent[0].age ) && ( units_difference < 1 ) && (amount_saved > 0) ||
(expensive[0].age > recent[0].age) && ( units_difference < 1 ) && (amount_saved > 0) )  )
                                             {  fprintf( shp, "  Cheapest: %s %s, purchased on %s for a unit price of $%3.3f %s
at %s. Savings: $%3.2f \n  Most Expensive: %s %s purchased at %s on %s for $%3.2f and $%3.3f %s.\n  PRICE INCREASE!  ",
                                    best[0].brand, best[0].name, best[0].date_of_purchase, best[0].price,
best[0].price_units_compare, best[0].store, amount_saved,
                                    expensive[0].brand, expensive[0].name, expensive[0].store, expensive[0].date_of_purchase,
expensive[0].totalpaid, expensive[0].price, expensive[0].price_units_compare); print_flag = YES; price_increase_detect = 1;
                                     fprintf( shp, "Your most recent purchase was %s %s, purchased on %s for a unit price\n
of $%3.3f %s at %s.\n\n",
                                       recent[0].brand, recent[0].name, recent[0].date_of_purchase, recent[0].price,
recent[0].price_units_compare, recent[0].store); }

                                    /* if above print statements were not printed, then print this one */
                                    if (price_increase_detect == 0 ) {
                                    if ( (best[0].price != expensive[0].price) && (best[0].price != 100) &&
(expensive[0].price != 0) && (amount_saved > 0) )
                                  {  fprintf( shp, "  Cheapest: %s %s, purchased on %s for a unit price of $%3.3f %s at %s.
Savings: $%3.2f \n  Most Expensive: %s %s purchased at %s on %s for $%3.2f and $%3.3f %s.\n",
                                    best[0].brand, best[0].name, best[0].date_of_purchase, best[0].price,
best[0].price_units_compare, best[0].store, amount_saved,
```

```
                                    expensive[0].brand, expensive[0].name, expensive[0].store, expensive[0].date_of_purchase,
expensive[0].totalpaid, expensive[0].price, expensive[0].price_units_compare); print_flag = YES;
                                      fprintf( shp, "  Your most recent purchase was %s %s, purchased on %s for a unit price
of $%3.3f %s at %s.\n\n",
                                        recent[0].brand, recent[0].name, recent[0].date_of_purchase, recent[0].price,
recent[0].price_units_compare, recent[0].store); } }

                                   if ( (best[0].price != expensive[0].price)  && (best[0].price != 100) &&
(expensive[0].price != 0) && (amount_saved == 0) )
                                  {  fprintf( shp, "  Cheapest: %s %s, purchased on %s for a unit price of $%3.3f %s at %s.
Savings: None\n  Most Expensive: %s %s purchased at %s on %s for $%3.2f and $%3.3f %s.\n  The better deal is to buy the item
with the $%3.3f %s unit price. You spend the same but get more.\n",
                                    best[0].brand, best[0].name, best[0].date_of_purchase, best[0].price,
best[0].price_units_compare, best[0].store, expensive[0].brand, expensive[0].name, expensive[0].store,
                                    expensive[0].date_of_purchase, expensive[0].totalpaid, expensive[0].price,
expensive[0].price_units_compare, best[0].price, best[0].price_units_compare ); print_flag = YES;
                                      fprintf( shp, "  Your most recent purchase was %s %s, purchased on %s for a unit
price\n  of $%3.3f %s at %s.\n\n",
                                        recent[0].brand, recent[0].name, recent[0].date_of_purchase, recent[0].price,
recent[0].price_units_compare, recent[0].store);  }

                                   if ( (best[0].price != expensive[0].price)  && (best[0].price != 100) &&
(expensive[0].price != 0) && (amount_saved < 0) )
                                  {  fprintf( shp, "  Cheapest: %s %s, purchased on %s for a unit price of $%3.2f and $%3.3f %s
at %s. Savings: $%3.2f \n  Most Expensive: %s %s purchased at %s on %s for $%3.2f and $%3.3f %s.\n",
                                    best[0].brand, best[0].name, best[0].date_of_purchase, best[0].totalpaid, best[0].price,
best[0].price_units_compare, best[0].store, amount_saved,
                                    expensive[0].brand, expensive[0].name, expensive[0].store, expensive[0].date_of_purchase,
expensive[0].totalpaid, expensive[0].price, expensive[0].price_units_compare );
                                    fprintf( shp, "  Your most recent purchase was %s %s, purchased on %s for a unit price\n
of $%3.3f %s at %s.",
                                        recent[0].brand, recent[0].name, recent[0].date_of_purchase, recent[0].price,
recent[0].price_units_compare, recent[0].store);
                                    if(amount_saved < 0) { fprintf( shp, " A negative savings means the cheaper per unit item
requires you purchase a LARGER amount for\n  an additional cost of $%3.2f more than expensive per unit item!\n  RealSave
Score is %2.0f out of 10.\n\n", negative_amount_saved, savings_index ); } print_flag = YES;  }
```

```
                                 if ( (best[0].price == expensive[0].price) && (amount_saved == 0 ) ) {
                                fprintf( shp, "  Price: %s %s costs $%3.2f. The unit price is $%3.3f %s and is the only
listed price.\n",
                                  best[0].brand, best[0].name, best[0].totalpaid, best[0].price,
best[0].price_units_compare); print_flag = YES;
                                fprintf( shp, "  Your most recent purchase of %s %s, was made on %s at %s.\n\n",
                                       recent[0].brand, recent[0].name, recent[0].date_of_purchase, recent[0].store); }


                               }  //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 an item is new we switch the statement to reflect that and we have to back up to [index - 1] to do
that */
                    if( STOP != YES )  {


                                             if ( ( print_flag == YES ) && ( print_force == NO ) ) { fprintf( shp,"  Next/New
item: %s\n\n", shop[index].name); print_flag = NO; }
                                               else  { if (index < count) fprintf( shp,"  %s is a NEW or EXPIRED item so no
pricing yet.\n\n", shop[index-1].name); print_force = YES; }
                                             if ( ( print_force == YES ) && ( index != count ) ) { fprintf( shp,"  Next/New
item: %s\n\n", shop[index].name); print_force = NO; }

                                             /* reset all of the storage arrays for highest, lowest price, and recent purchase
*/
                                             strcpy(best[0].brand, ""); strcpy(best[0].name, "");
                                             strcpy (best[0].store, ""); strcpy(best[0].date_of_purchase, "");
                                    strcpy(best[0].price_units_compare, "");
                                    best[0].price = 100.00;

                                    strcpy(expensive[0].brand, ""); strcpy(expensive[0].name, "");
                                             strcpy (expensive[0].store, ""); strcpy(expensive[0].date_of_purchase, "");
                                    strcpy(expensive[0].price_units_compare, "");
```

```
                                    expensive[0].price = 0.00;

                                    price_increase_detect = 0;

                                    strcpy(recent[0].brand, ""); strcpy(recent[0].name, "");
                                            strcpy (recent[0].store, ""); strcpy(recent[0].date_of_purchase, "");
                                    strcpy(recent[0].price_units_compare, "");
                                    recent[0].age = 500;

                                  } //if STOP != YES

                                } //if match !=0 and index <= count


                        /* if at the end of the shopping list then break from for loop */
                                if ( ( index == count ) && (print_flag == NO) ) { fprintf( shp,"  %s is a NEW or EXPIRED item so no
pricing yet.\n\n", shop[index-1].name);
                                   break;  }

                                /* reset and continue to next item in linked list */
                                info = info->next;
                                    Match = -1;
                                        matching_city = -1;
                                print_flag = NO;

                            } //end for loop

                    /* print summary */
                    fprintf( shp, "\n             List complete. You have selected %d items to shop for. ", count );
                    fprintf( shp, "\n        Your estimated cost for groceries today excluding NEW items is $%3.2f \n\n",
total_shopping_price );


                printf("\n        Done building list. Your shopping list is 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:  gram_ounces_gram();
               break;
             case 9:  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");
  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.  Grams->Ounces->Grams\n");
  printf("                        9.  Exit\n");

    do  {
    printf("\n               Enter your choice:   ");
    gets(s);
    c = atoi(s);
   }//do
  while ( c < 0 || c > 9 );
  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();
}

gram_ounces_gram() {
    int select;
        float grams, o;
        clrscr();
        printf("        Select 1 for converting grams. or 0 for converting ounces    ");
        scanf("   %d", &select);
          if( select == 0 ) { printf("\n                        Enter the ounces to convert:    ");
                    scanf(" %f", &o); grams = o * 28.34952; printf("\n                          %3.2f ounces = %3.2f grams",
o, grams); }
      if( select == 1 ) {printf("\n                       Enter the grams to convert:   ");
                    scanf(" %f", &grams); o = grams * .03527396; printf("\n                          %3.2f grams = %3.2f
ounces", grams, 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;


        /* 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:  \"shopdata\"\n   Date: %s \n", TodaysDate );
        fprintf(makelist, "   PPU, BRAND, DOP, Store, PD, ITEM\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  ) {
                  /* save this list to a file */
                  if (strlen( info->item_Brand ) < 4) {
                  fprintf(makelist,"  %3.3f    \t%s   \t\t%s    \t\t%s \tpaid $%3.2f   \t%s\n", info->item_UnitCost, info-
>item_Brand,
                    info->item_DatePurchased, info->item_storeName, info->item_actualprice_paid, info->item_Name ); }
           if (strlen( info->item_Brand ) == 4) {
                  fprintf(makelist,"  %3.3f    \t%s   \t\t%s   \t\t%s   \tpaid $%3.2f   \t%s\n", info->item_UnitCost, info-
>item_Brand,
                    info->item_DatePurchased, info->item_storeName, info->item_actualprice_paid, info->item_Name ); }
                  if (strlen( info->item_Brand ) == 5) {
                  fprintf(makelist,"  %3.3f    \t%s   \t%s   \t\t%s   \tpaid $%3.2f   \t%s\n", info->item_UnitCost, info-
>item_Brand,
                    info->item_DatePurchased, info->item_storeName, info->item_actualprice_paid, info->item_Name ); }
                  if (strlen( info->item_Brand ) > 5) {
                   fprintf(makelist,"  %3.3f \t%s   \t%s          \t%s  \tpaid $%3.2f   \t%s\n", info->item_UnitCost, info-
>item_Brand,
                    info->item_DatePurchased, info->item_storeName, info->item_actualprice_paid, info->item_Name ); }
                  } //if strcmp()
                count += 1;
                info = info->next;
           } // end while
    /* print screen message and print the other to the file itself */
    printf("\n               File: %s.dat created. Press any key to go to menu", TodaysDate);
```

```
    fprintf(makelist, "\n\n              There are currently %d items in the grocery list database", count);
    fclose( makelist );
        getch();

        }  //end list


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

        fp = fopen( "shopdata",  "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 shopdata..." );

        }       // end save


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

         fp = fopen( "shopdata", "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: shopdata - press a key to continue" );
      getch();

     } // end load

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

        fp = fopen( "shopdata.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 shopdata.bak - press a key to continue" );
        getch();

    } // end saveDataBackup
```