```
/* Modified by Josheta Srinivasan */
#include "party.h"
/*
 * This function allows a nparty-length party to be reordered arbitrarily,
 * without allowing character duplication!
 */
void reorderParty(unsigned int nparty)
{
#if 0
   reorderPartyRCW(nparty);
#else
   /* Need to shuffle: 1. char** PRT_nam => array of strings of names
    *                  2. int** PRT_cds => array of cd arrays
    *                  3. float* PRT_gld => array of gold floats
    *                  4. float* PRT_hgt => array of height floats
    *                  5. item_t** PRT_pck => array of item arrays
    *                  6. item_t*** PRT_eqp => array of array of eqItem arr
    */

   /* declare Vars */
   int i, j;                  /* for looping */
   int* iArr;                    /* index array */
   char** PRT_namN;              /* shuffled array of names */
   unsigned int** PRT_cdsN;      /* shuffled array of cds */
   float* PRT_gldN;              /* shuffled array of golds */
   float* PRT_hgtN;              /* shuffled array of heights */
   item_t** PRT_pckN;            /* shuffled array of backpack items */
   item_t*** PRT_eqpN;           /* shuffled array of equipped items */
   char choice, choice2;
   char* AA, *AA_2;
   int nUnord = nparty;          /* number of characters not ordered by user */
   int kpress = 0;
   int* iUnord;                /* indexes that havent been ordered */
   int* orgData;                 /* keeps the original data order */
   void delElem(int** arr, int n, int pos);
   void delChar(char** arr, int n, int pos);

   /* Malloc an array of index; use this to track the ordering */
   iArr = malloc(nparty*sizeof(int));
   assert(iArr);
   for(i=0; i<nparty; i++)      /* enumerate array */
      iArr[i] = i;

   /* Malloc an array of indexes: use this to track which chars have been ordered
*/
   iUnord = malloc(nparty*sizeof(int));
   assert(iUnord);
   for(i=0; i<nparty; i++)      /* enumerate array */
      iUnord[i] = i;

   /* Malloc an array of indexes: use this to remember original order */
   orgData = malloc(nparty*sizeof(int));
   assert(orgData);
   for(i=0; i<nparty; i++)      /* enumerate array */
      orgData[i] = i;

   /* Malloc new arrays for each array needed to shuffle */
      /* Char name array */
   PRT_namN = malloc(nparty*sizeof(char*));
```

```
assert(PRT_namN);
  /* CDs array */
PRT_cdsN = malloc(nparty*sizeof(int*));
assert(PRT_cdsN);
for(i=0; i<nparty; i++)
{
   PRT_cdsN[i] = malloc(CD_LEN*sizeof(int));
   assert(PRT_cds);
}
   /* Gold array */
PRT_gldN = malloc(nparty*sizeof(float));
assert(PRT_gldN);
  /* Height array */
PRT_hgtN = malloc(nparty*sizeof(float));
assert(PRT_hgtN);
  /* backpack items  array */
PRT_pckN = malloc(nparty*sizeof(item_t*));
assert(PRT_pckN);
  /* eqp items array */
PRT_eqpN = malloc(nparty*sizeof(item_t**));
assert(PRT_eqpN);
for (i=0; i<nparty; i++)
{
   PRT_eqpN[i] = malloc(EQP_SLOTS*sizeof(item_t*));
}


/* Malloc an array of allowed responses */
AA = malloc((nparty+1)*sizeof(char));
assert(AA);
for(i=1; i<(nparty+1); i++)
   AA[i] = '0' + (i-1);
AA[0] = 'k';

AA_2 = malloc(2*sizeof(char));
assert(AA_2);
AA_2[0] = 'K';
AA_2[1] = 'R';

/* Print original party order */
printf("\nORIGINAL PARTY ORDER IS: \n");
printParty(nparty, 1); /* Prints party in detail */
printf("\n");


/* Prompt and reorder */
for(i=0; i<nparty-1; i++)
{
   /* PROMPT */
   printf("What stalwart hero shall stand at rank #%d\n", i);
   printf("    [K]eep original ordering, abandoning any reorder in progress\n");
   for(j=0; j<(nUnord); j++)
   {
      printf("    [%d] %s\n", iUnord[j],PRT_nam[iUnord[j]]);
   }
   /* GET CHOICE */
   choice = getCharInSet("Your choice: ", nUnord+1, AA);

   /* REORDER */
```

```
      if(choice == 'k' || choice == 'K')
      {
         kpress = 1;
         goto END_REORD;
      }

      else
      {
         /* convert choice to integer: choice - '0' */
         iArr[i] = choice - '0'; /* assign iarr[i] to the chosen char */

         /* delete chosen element */
         delElem(&iUnord, nUnord, orgData[iArr[i]]);

         /* remove choice element from AA */
         delChar(&AA, nUnord, orgData[iArr[i]]);

         /* reduce nUnord by 1 */
         nUnord --;

      }

   }

  iArr[i] = AA[1] - '0'; /* remaining slot allocated */

END_REORD:

   if(kpress!=1)
   {
      /* Print original party order */
      printf("\nORIGINAL PARTY ORDER IS: \n");
      printParty(nparty, 1); /* Prints party in detail */
      printf("\n");

      /* Print new order */
      printf("YOUR NEW ORDER IS: \n");
      for(i=0; i<nparty; i++)
      {
         printf("    [%d] %s\n", orgData[i],PRT_nam[iArr[i]]);
      }


      printf("[K]eep original, [R]eorder: ");
      choice2 = getCharInSet("", 2, AA_2);
      switch (choice2)
      {
         case 'k':
         case 'K':
            for(i=0; i<nparty; i++)
               iArr[i] = orgData[i];
            break;
         case 'r':
         case 'R':
            break;
      }

     /* fill out each array with shuffled values (from index array ) */
      for(i=0; i<nparty; i++)
```

```
      {
         PRT_namN[i] = PRT_nam[iArr[i]];
         for(j=0; j<CD_LEN; j++)
            PRT_cdsN[i][j] = PRT_cds[iArr[i]][j];
         PRT_gldN[i] = PRT_gld[iArr[i]];
         PRT_hgtN[i] = PRT_hgt[iArr[i]];
         PRT_pckN[i] = PRT_pck[iArr[i]];
         for(j=0; j<EQP_SLOTS; j++)
            PRT_eqpN[i][j] = PRT_eqp[iArr[i]][j];
      }


      /* copy the shuffled arrays into new arrays */
      for(i=0; i<nparty; i++)
      {
         PRT_nam[i] = PRT_namN[i];
         for(j=0; j<CD_LEN; j++)
            PRT_cds[i][j] = PRT_cdsN[i][j];
         PRT_gld[i] = PRT_gldN[i];
         PRT_hgt[i] = PRT_hgtN[i];
         PRT_pck[i] = PRT_pckN[i];
         for(j=0; j<EQP_SLOTS; j++)
            PRT_eqp[i][j] = PRT_eqpN[i][j];

      }

   }


   /* free the malloced arrays */
   free(PRT_namN);
   for(i=0; i<nparty; i++)
   {
      free(PRT_eqpN[i]);
      free(PRT_cdsN[i]);
   }
   free(PRT_cdsN);
   free(PRT_eqpN);
   free(PRT_gldN);
   free(PRT_hgtN);
   free(PRT_pckN);
   free(iUnord);
   free(orgData);
   free(AA);
   free(AA_2);

   /* free the index array */
   free(iArr);

#endif
}

void delElem(int** arr, int n, int pos)
/* takes in an inetegr array and deletes the element index pos */
{
   /* vars */
   int i;
   int* arrDel = *arr;  /* array to manipulate */
```

```
   assert(pos>=0 && pos<n);

   /* move all from pos up */
   for(i=pos; i<n-1; i++)
      arrDel[i] = arrDel[i+1];

}

void delChar(char** arr, int n, int pos)
/* takes in a char array and deletes the element index pos */
{
   /* vars */
   int i;
   char* arrDel = *arr;  /* array to manipulate */

   assert(pos>=0 && pos<n);

   /* move all from pos up */
   for(i=pos; i<n-1; i++)
      arrDel[i] = arrDel[i+1];

}
```