Section Solution

Solution 1: Sparse String Arrays

a)

```
static void StringFree(void *elem) { free(*(char **) elem); }
void SSANew(sparsestringarray *ssa, int arrayLength, int groupSize)
{
   ssa->arrayLength = arrayLength;
   ssa->groupSize = groupSize;
   ssa->numGroups = (arrayLength - 1)/groupSize + 1;
   ssa->groups = malloc(ssa->numGroups * sizeof(group));

   for (int i = 0; i < ssa->numGroups; i++) {
      ssa->groups[i].bitmap = malloc(groupSize * sizeof(bool));
      bzero(ssa->groups[i].bitmap, groupSize * sizeof(bool));
      VectorNew(&ssa->groups[i].strings, sizeof(char *), StringFree, 1);
   }
}

void SSADispose(sparsestringarray *ssa)
{
   for (int i = 0; i < ssa->numGroups; i++) {
      free(ssa->groups[i].bitmap);
      VectorDispose(&ssa->groups[i].strings);
   }

   free(ssa->groups);
}
```

b)

```
bool SSAInsert(sparsestringarray *ssa, int index, const char *str)
{
   int grp = index / ssa->groupSize;
   int indexWithinBitmap = index % ssa->groupSize;
   int indexWithinVector = 0;

   for (int i = 0; i < indexWithinBitmap; i++) {
      if (ssa->groups[grp].bitmap[i])
         indexWithinVector++;
   }

   const char *copy = strdup(str);
   bool previouslyInserted = ssa->groups[grp].bitmap[indexWithinBitmap];
   if (previouslyInserted)
      VectorReplace(&ssa->groups[grp].strings, &copy, indexWithinVector);
   else
      VectorInsert(&ssa->groups[grp].strings, &copy, indexWithinVector);
   ssa->groups[grp].bitmap[indexWithinBitmap] = true;

   return !previouslyInserted;
}
```

c)

```
typedef void (*SSAMapFunction)(int index, const char *str, void *auxData)
void SSAMap(sparsestringarray *ssa, SSAMapFunction mapfn, void *auxData)
{
   int index = 0;
   for (int i = 0; i < ssa->numGroups; i++) {
      group *grp = &ssa->groups[i];
      int groupSize = ssa->groupSize;
      if (i == ssa->numGroups - 1 && ssa->arrayLength % ssa->groupSize > 0)
         groupSize = ssa->arrayLength % ssa->groupSize;

      int indexOfNonEmptyString = 0;
      for (int j = 0; j < groupSize; j++) {
         const char *str = "";
         if (grp->bitmap[j]) {
            str = *(char **) VectorNth(&grp->strings, indexOfNonEmptyString);
            indexOfNonEmptyString++;
         }
         mapfn(index, str, auxData);
         index++;
      }
   }
}
```

Solution 2: Serializing Lists of Packed Character Nodes

```
int *serializeList(const void *list)
{
   int *serialization = malloc(sizeof(int));
   int serializationLength = sizeof(int);
   const void **curr = (const void **) list;
   int numNodes = 0;

   while (curr != NULL) {
      const char *str = (const char *)(curr + 1);
      serialization = realloc(serialization,
                              serializationLength + strlen(str) + 1);
      strcpy((char *) serialization + serializationLength, str);
      serializationLength += strlen(str) + 1;
      curr = (const void **) *curr;
      numNodes++;
   }

   *serialization = numNodes;
   return serialization;
}
```

Solution 3: The C `multitable`

a)

```
typedef struct {
   hashset mappings;
   int keySize;
   int valueSize;
} multitable;

void MultiTableNew(multitable *mt, int keySizeInBytes, int valueSizeInBytes,
                   int numBuckets, MultiTableHashFunction hash,
                   MultiTableCompareFunction compare)
{
   mt->keySize = keySizeInBytes;
   mt->valueSize = valueSizeInBytes;
   HashSetNew(&mt->mappings, keySizeInBytes + sizeof(vector), numBuckets,
              hash, compare, NULL);
}
```

b)

```
void MultiTableEnter(multitable *mt, const void *keyAddr, const void *valueAddr)
{
   char buffer[mt->keySize + sizeof(vector)];
   vector *values;
   void *found = HashSetLookup(&mt->mappings, keyAddr);
   if (found == NULL) {
      memcpy(buffer, keyAddr, mt->keySize);
      values = (vector *)(buffer + mt->keySize);
      VectorNew(values, mt->valueSize, NULL, 0);
      VectorAppend(values, valueAddr);
      HashSetEnter(&mt->mappings, buffer);
   } else {
      values = (vector *)((char *) found + mt->keySize);
      VectorAppend(values, valueAddr);
   }
}
```

c)

```
typedef struct {
   MultiTableMapFunction map;
   void *auxData;
   int keySize;
} maphelper;
```

```
static void HashSetMapper(void *elem, void *auxData)
{
   int i;
   maphelper *helper = auxData;
   vector *values = (vector *)((char *) elem + helper->keySize);
   for (i = 0; i < VectorLength(values); i++)
      helper->map(elem, VectorNth(values, i), helper->auxData);
}

void MultiTableMap(multitable *mt, MultiTableMapFunction map,
                   void *auxData)
{
   maphelper helper = {map, auxData, mt->keySize};
   HashSetMap(&mt->mappings, HashSetMapper, &helper);
}
```

Solution 4: `multitable` Client Code

```
void ListRecordsInRange(multitable *zipCodes, char *low, char *high)
{
   char *endpoints[] = {low, high};
   MultiTableMap(zipCodes, InRangePrint, endpoints);
}

static void InRangePrint(void *keyAddr, void *valueAddr, void *auxData)
{
   char *zipcode = (char *) keyAddr;
   char *city = *(char **) valueAddr;
   char **endpoints = (char **) auxData;
   char *low = endpoints[0];
   char *high = endpoints[1];

   if ((strcmp(zipcode, low) >= 0) && (strcmp(zipcode, high) <= 0))
      printf("%5s: %s\n", zipcode, city);
}
```