Data structures

Assignment 3: Fun with binary files.

Group 1292
Team Number 2
Pablo Almarza Marques
Jorge González Gómez

Table.

We started working and we realised that what we were doing was more difficult than the actual practice, since we were making it so the table was loaded in RAM, which was wrong.
There were three functions where we had to do something that required a second thought. First, table_open, where we saved num_records into RAM memory just to avoid reading the file every time a record is inserted. Also we save in the the structure the position of the record and the position of the first record.
Second, table_read_record, that was the most difficult one, so at the beginning we check if the record that we are writing has memory allocated, if not, we allocate it.
Then, we write in this way:

```
for (i = 0; i < t->ncols; i++) {
      switch (t->types[i]) {
         case STR:
            t->current_record[i] = strdup(aux);
            aux += strlen(aux) + 1;
            continue;
         case INT:
            num = (int*) malloc(sizeof(int));
            *num = *((int*) aux);
            t->current_record[i] = num;
            aux += sizeof(int);
            continue;
         case LLNG:
            bignum = (long long*) malloc(sizeof(long long));
            *bignum = *((long long*) aux);
            t->current_record[i] = bignum;
            aux += sizeof(long long);
            continue;
         case DBL:
            dbl = (double*) malloc(sizeof(double));
            *dbl = *((double*) aux);
            t->current_record[i] = dbl;
            aux += sizeof(double);
            continue;
         default:
            fprintf(stderr, "Invalid type for column in read in column %d\n", i + 1);
            break;
      }
   }
```

We allocate the necessary memory for each type that we will copy and then, copy it.

Finally, table_insert_record. To do it, before writing, we calculated the total length of the record, then write it, and finally writing each row of the record. For the string case we did this:

```
case STR:
        fwrite(values[i], sizeof(char), strlen(values[i]), t->f);
        fwrite(&null, sizeof(char), 1, t->f);
```

And we declared that null as char null ='\0'. This was done in order to write the null terminating character since we're dealing with char* and not fixed-length arrays. At the end of the file always add one to the number of records and seek the first record to finally write the number of records.

Index.

The index part was more difficult than the table's, since it was more complex than the table's. Our first problem was to decide how the structure would be in index, we decided to go with two structures, one for the indexes, and the other for the data of the indexes that held the key. We followed the implementation given by the PDF, since for us it was easier to follow it than creating our own from zero. For the binary search, we used the one provided by C.
Some problems that we had were with the function indes_save, since when we started doing it we realised that we didn't have the path, so, instead of keeping the file open, we saved the path in the structure when doing index_open, and then re-opened the file in index_save.
Another problem was in index_put, because we were doing invalid frees, but that was quickly fixed by changing the position of the freeing codeblock to its correct place.
Overall, the most interesting part is our bsearch implementation. We used the C's provided bsearch function, which returned a pointer to our desired key. Then, we simply used that value using the de-reference symbol (asterisk) and it was mostly easy.
For instance, to find and use the key we used the following:

```
idxTemp = bsearch(&key, idx->idx_datas, idx->nidx,
sizeof(idx_data), idx_compare);
/* If the key is found... */
if (idxTemp) {
    arrIdx = (*idxTemp).nrec;
    (*idxTemp).records; // realloc this;
     [...]  //error checking
        (*idxTemp).records[arrIdx] = (unsigned long) pos;
        (*idxTemp).nrec++;
        return idx->nidx;
    }
```

Another thing to mention is that we needed the type for the compare function, so we had to pass it as a global variable. It's no big deal, since we're just using the type INT. It also can be avoided, but it would have helped if we decided to add more types to the indexes.

In the end, we learned a lot about how to deal with binary staff and very low level code in C, which will surely help us in the future.