Group Programming Project Design Document

Dual Heap Replacement Selection Sort

By: Jared Rickert, George Velikson, Kelden Mayer, Chris Vadner, and Sanju Neupane

**Introduction:**

This program was designed to perform a replacement selection sorting technique using a dual heap abstract data type. The program reads in a file of unsorted records and uses sorting techniques and algorithms to sort output the records into a new file that consists of the records in a sorted form.

The program highlights the use of the replacement selection sorting technique, the k-way tournament sort, and then finally a merge to combine the data into one.

When the program is finished, there is also a log file that is generated that includes statistics retrieved from the program’s runs.

**Data Structures:**

To begin listing the data structures in the program, there is a **const char** DELIM which acts as a delimiter for fields in the program. The delimiter in this program is set to “,” to serve its purpose. As for separating the records, those are divided by a new line. These are important because the delimiters make it possible to input the data record by record.

Next, in the header file “collection.h”, there are four important classes as follows: **class** Collection, **class** Field, **class** Index, and **class** Record. These all serve different purposes. The Collection class is the main class in the program that accepts different data types and contains the functions needed to perform the main sorting and merging algorithms. Next, the “Field” class contains the necessary functions required to compare fields and give instructions on what to do if an error arises in the process. The class named “Index” contains the functions that make it possible for the program to have an index. Lastly, the Record class has the functions in it that allow the program to interact and perform actions on records in given files.

The next set of data structures in this program are in the form of the **struct** data structure. There are two major structs used in this program. They are **struct** Key and **struct** Field. The *Key struct* contains information including the types **const enum Type** INTEGERand STRING**.** Inside the struct, there is a **const size\_t** pos which is for the position of the key in the record and a **const string** name which holds the name of the key. The key struct is a metadata object that holds information about if a field is in a record. The next struct in the program is called *Field*. First off, this struct contains a **const union** Data data type and unions the **int** *integer* and the **string** *pointer* pointing to the variable *string*. From this, there emerges a variable named *data*. There is also a **Key** with the variable name of *key*. In the struct there are also several functions that consist of true or false operators used to compare Fields of data.

There were also two **typedef** operations used. These were used to make it easier to keep track of different data types. The first one was **typedef size\_t Word** which made the variable Word act as the data type size\_t. The next one was **typedef string key\_pair\_t** which establishes key\_pair\_t as a string data type that is used for easier insertion of keys. Both of these variables are used multiple times in the program and are essential to its function.

In the program, there included two important **string** data types in the code. The string **file\_name** is self-explanatory. It simply holds the name of file being read from in the variable. This string is also used as an unchangeable **const string** for other purposes. Like the previous string listed, the string **output\_file** holds the name of the file that the results of the program are being output to.

The data type **size\_t** was especially important and generously used in this program. The first occurrence of it came in the variable **i**. This variable indicates position in accordance to the offset of a record. The constant variable **offset** is used to return the number of bytes that are offset in a record. There is also another size\_t variable that is also a const static variable. This is called **HEAP\_SIZE** and it holds the value of how large the heap can be in the sorting process. The last size\_t variable that was used was called **pos**. This simply kept track of the positioning of which records are being compared and sorted.

It was essential to include many types of stream types in the program. The variables that held the traits of this data type were **stream,**  **input, buffer, offsets, input,** and **output**. The main purpose of these were to be able to read and write to different files and have the ability to transfer data along an I/O stream. They are important to have because their duties included reading from the file containing the records, outputting the file of sorted records, and storing the logs of the program.

**Functions:**

In this program, there were many functions that made it work and perform essential operations. To start off, there were many functions that were implemented from the class Collection. One function is one called **Collection()** that serves as a constructor. This constructor also opens the file and initiates both the keys and index of the program. Complementing that, there is a destructor called **~Collection()** that terminates the process and closes the file that is currently opened. The function names **init\_file()** was designed to emit an error message if there is a problem opening the file. Similar to that, there is a function **init\_keys()** which also performs an error checking method to be sure the keys are valid and in working order along with the function **init\_index()** which works relatively the same.

An extremely important and vital function in this program is called **parse\_key()**. The function first starts by inputting the string passed to it and then takes away the delimiter (,). It then determines what the data type of the key is. Once it determines the data type, it changes the key to that desired type. If the key is not either an integer or a string, the function will return an error message.

There were also numerous functions that were implemented from the Record class. The first function is simply the constructor **Record()**. This just initializes things. The next function is **get()** which obtains the key that is being used. It does this by checking the delimiters being used and returns with a desirable key.

There are three Boolean functions that are used that compared the keys in the program. They were **lte()** which determined if one was “less than or equal to” the other, **gt()** which determined if one was “greater than” the other, and **gte()** which determined if one was “greater than or equal to” the other. This was important when performing the sorting actions of the program.

The field class also had several important functions. The function **string()** checks if the given key is a string when comparing. If it isn’t, it returns an error message. Otherwise, the program proceeds accordingly. There are also several Boolean comparison functions. These include if fields are greater than, less than, or equal to each other. It will perform necessary actions to reorder them as desired. The field class also has a destructor called **~Field()**. If there is a problem when comparing two fields, it will trigger the **panic\_on\_bad\_compare()** function. This function will display an error and crash the program.

One of the most essential functions in the program is called **heapify()**. This make the current desired keys into a heap and the program will continue.

Another extremely important function is called **sort()**. This function correctly and efficiently calls other functions to perform the correct sorting algorithms to sort the keys. Another function that helps with this operation is called **replacement\_selection\_sort()** which performs the replacement selection sorting algorithm to sort through the records. After those are done, the **kway\_merge()** takes place and finishes the process before outputting to a sorted file of records.

**The Main Program:**

In greater depth, this program performs a replacement selection sorting technique using a dual heap abstract data type. The program reads in a file containing anywhere from a small to an extremely large amount of unsorted records. The records consist of different data types including integer-valued keys, string-valued keys, and ordered pairs of the two. Next, the algorithms run, and the sorting technique is performed. As the program runs, a replacement selection sort takes place. Shortly after, the k-way tournament sort takes place and is finally followed by a complete merge of the data. This is done recursively until the records are completely sorted. When the algorithm reaches the end, it means the records are sorted and are placed in a new file that consists solely of the sorted records.

There is also a logging file that stores the statistics of the sorting algorithm’s runs. These statistics include the name of the file being read from and the name of the file being written to. Also, the number of records that can fit in memory along with the number of records, runs, and information regarding the size of the records. These statistics can be found in the logging file that is created when the program is completed.