Program3

Ching-Wei Lin

#987563037

Debugger:

This program is very similar to the last one. I used the abstract data type for the data type which can be assigned to different derived classes. Then, here come the differences, this time I implement the 2-3 tree as my data structure. Moreover, there are two different kinds of 2-3 tree, which one is for storing avatar’s characters and the other is for avatar’s tools. They are almost the same, but the data type they are storing. I used debugger for my data structure. This is my first time implementing a balance tree. It is much harder and more complicated than I thought. Because the data will be moved around the tree due to the shape of the tree, so many seg fault or memory leaks appear during the “moving process “. For example, when I am adding a data into a node which already has two data, I have to draw up the middle one and split rest of the data into two children of that node. Therefore, I have to create a new node if there is no parent there or I have to create a new node for the new child of the node. I have to allocate the data before I store it in to avoid seg faults.

Analysis:

At the beginning of building the program, I spent a lot of time implement the abstract data types because there are up to eight different classes and each of them has three functions to implement differently. Then, I realized that abstract data type is very convenient to use but it takes time to build up the foundation. The three functions are add, display and remove. They are not implemented in the basic class and the implementation for eight different classes actually are very similar, though I just have to implement them. I think there is might be a way to implement them only once or twice just for different data types, but I have not figure it out yet.

Then, the second part of the program is build the 2-3 tree. I built the add function first which is super complicated. For my add function, I traverse to the leaves and then check if there are already two data in the node. If no, it is perfect, I can just store the data into the node before I check which one is bigger and set them to the correct place. If yes, I have to look up to its parent and see if there is a space for the data and keep checking the space until there is a one and then split the data into the children correctly. Also, after the space checking, I have to compare which one is the biggest one, middle one and smallest one to determine the right place them should be. Besides, there are two kinds of 2-3 tree to be implemented, which is for characters and the other is for tools, so I have to build the tree twice. The display function and the remove function are much easier because there’s no need to display in order. I have only make sure to display it all and remove it all recursively.

The third part of the program is operator overloading. At the beginning, I don’t even know why should I need to use operator overloading because I could build it well without it. Then, I seemed it as a way to simplify my code, which each of the operator represents a kind of function. I can use it as a function call or use it without thinking about what data type is the object. It is a powerful method. If I have more time to do the assignment, I will make more use of it.