Mauricio Deguchi

UID: 405582562

CS 31

February 2, 2022

**Project 3**

**CDPlayer**

*Challenges:*

The biggest challenge for this project was the initial direction. Once I got going, the objective of the program became more and more intuitive for me. The major turning point in my understanding of the program was once I realized flags would be extremely useful. Once I had the mindset of placing flags, the rest of the program fell into place nicely. One major issue I struggled with was understanding the error produced when I fed my code an empty string. I began by checking if the string was valid by verifying that the first letter was an ‘O’, if not we wouldn’t proceed checking. This check would return an error for an empty string, so I had to separately include a check to ensure that we were not for and empty string. The other major issue was understanding how opening the player would pause the player (it required a play to continue numbers after opening while playing). For this, I started a new flag which would check how the player was stopped, either by stop or by opening the player.

*Code Structure:*

**isValidCDPlayerString**

For starters, I began with the “isValidCDPlayerString” function. This function begins by declaring various flags: (i) the overarching flag checking if the string is valid, which by default is set to true (flag), (ii) the isOpen flag would be toggled when we open and close the player, which by default was false, (iii) the isCDin flag would be toggled between in and out when we insert and remove the CD, by default this is set to false, and lastly (iv) the isPlaying keeps track of whether the CD player is playing, by default this is set to false. Aside from the flags, I also used one more variable which is the integer trackCounter. This value is initially 0, since no CD has been inserted.

We then begin parsing the string by first, ensuring that an empty string is not fed in. If not, then we verify that the first letter of the string is an ‘O’ to begin our loop (if not, then our system ends and returns false). We then move into the for loop to extract one letter of the string at a time. The loop goes through each of the positions in the string, where at each step in the loop we analyze only one char. We then move into a switch statement containing all the possible valid characters (O, I, R, C, P, S, 1, 2, 3, 4, 5, 6, 7, 8 and 9) that can be included in a valid CDstring as cases, with the default of the switch setting our overall flag to false (in the case where an invalid character is in the string).

Let’s break down each of the 15 valid cases; each of these cases contains a near identical if-else statement. The if statement checks if the condition for landing on a given character is valid; if so, we will perform some adjustments to flags or counters. Otherwise, we will turn our overall flag to false since the conditions were not valid. O, C, R, P, and S are nearly identical. For O, we check if the player is closed, if so, we can set our flag isOpen to true and proceed. For R, we check if the player is open (isOpen == true) and there is a CD inside (isCDin == true), if so we remove the disk and proceed (isCDin = false). The only letter with a slight modification is I; I also starts our trackCounter to 1. Each time we insert the disk, the track counter resets to 1. Lastly, the numbers have identical structure between them; we check if our track counter equals the number of our string, if so, we play that track and move to the next one (increase trackCounter by 1). This was the only place I was unsure of, my code continue from 9 as if it were to expect a 10th track but has no way to accept an input of a 10th track.

If our string passes through all of these cases without falling into an else statement, we return the value of true, and our string is deemed valid.

**isOpen, hasCD, totalTracksPlayed, currentTrack**

These functions all take many details from the original function. Again, we loop through the string in the same manner as before, however, we now reduce our switch cases to only the characteristic we desire. Each of these functions call isValidCDPlayerString to verify that our string will work properly; if the string is valid, we move on to check our desired value. For example, for isOpen, we only have the case O and C, since we only seek to determine the final state of switches between these two states. Our flag is switch each time we encounter one of these chars and we output the final state of our flag. hasCD functions in an identical manner. totalTracksPlayed has all the numerical cases and adds one to a counter for each case; it then outputs this counter to main. Latsly currentTrack switches the value of a single integer each time a number is extracted from the string to the next value; the output is the last integer we extracted from the string +1 to show that we played that track and stopped at the next one without playing it.

*Test cases:*

**Valid:**

* OICP123SP456SORICP123 (verifies that the track counter returns 9)
* OIRICP123SOR (verifies that the current track is -1 since there is no CD)
* OICP123456789 (verified that current track goes to 10)
* OICOCORCOICPSOCP (string that doesn’t play a track, all ‘actions’ work though)

**Invalid:**

* xyz (checking invalid characters)
* oic (checking lowercase is rejected)
* OIC xyz (checking how we interpret spaces)
* P123S (does not start with O)
* OCIP123S (to have a cd inserted, the player must be opened first)
* OICR (to have a cd removed, the player must be opened first)
* OCOICC (player can only be closed if it is currently open)
* OCOICOO (player can only be opened if it is currently closed)
* OIP123 (to be played, a cd must be inserted, and the player closed)
* OCP123 (to be played, a cd must be inserted, and the player closed)
* OICS (to be stopped, a cd must be currently being played)
* OICP0123 (0 is not a valid track)
* OICP1245 (tack comes out of order)

\*Note: I tested all of the cases provided on the discussion board and all cases passed, but that is too many to include in this report.