Algorithm for Matching

We first read the CSV file line by line into the datalist. However, one thing to note here is that if there are line breaks in the greetings when filling out the form, the text after the line break will be independently included as a new line in the CSV file. Therefore, we determine whether a line is a data entry based on whether it starts with "2021/". If it is not a data entry, we merge it into the greetings of the previous entry.

If there are duplicate emails, we need to remove the other entries and keep only the last one. So, we reverse the datalist that we just organized and put the first data entry into the cleanlist. Then, we run a double-loop to check each data entry in the datalist to see if it already appears in the cleanlist. If it has, we add it to the repeat_list. If it does not appear, it means there are no duplicates, so we add it to the cleanlist.

We import the shuffle function from the random library because we need to randomize the order of the responses to ensure fairness in the matching process. Then, we group the data into four lists according to sexual orientation: heterosexual males, heterosexual females, homosexual males, and homosexual females.

```
procedure Match(groupA, groupB, couplelist)：
    for i (0 to 151) do
        foreach e(groupA) do
            foreach e(groupB) do
                if e(groupB) == matched or e(groupA) == matched then
                    skip this round
                elif the mail of e(groupB) == e(groupA) then
                    skip this round
                let total_gap be 0
                for j (5 to 30) do
                    total_gap <- total_gap + the abs of (e(groupA)[j]-e(groupB)[j])
                if total_gap == i then
                    couplelist append [e(groupA),e(groupB),total_gap]
                    let e(groupA) be matched
                    let e(groupB) be matched
                    end foreach
            end foreach
        end foreach
    if groupA == groupB then
        foreach e(groupA) do
            if e(groupA) != matched then
                left append e(groupA)
    return couplelist
end procedure
```

The above code is the core of this project. We have written a matching function with three parameters: 'alist', 'blist', and 'couplelist' to store the results. 'alist' and 'blist' contain the lists we sorted earlier, and we need to match each person in 'alist' with everyone in 'blist'. Therefore, the length of 'alist' must not exceed that of 'blist', so that we won't have people in 'alist' who have been paired up while others have no match. The outermost loop of this function runs from 0 to 150 because we want to select combinations with a total difference of 0 to 150. The second and third loops are used to match each person in 'alist' with everyone in 'blist', while the fourth loop calculates the absolute difference between the answers of the two people and adds them up. If the total difference equals the value of i (initially set to 0), we consider them a match and put their nicknames, gender, email addresses, and greetings into a list, which is then added to 'couplelist'. We also mark them as 'matched' in their original data. If 'alist' is equal to 'blist', it means that we have same-sex male couples or remaining heterosexual male couples. We keep any remaining people in 'final_left', with at most one person per group. We manually match them at the end.

Next, we split the couples into groups because Gmail can only send 500 emails per day. We package 245 pairs into one group, which is 490 emails. After importing some functions for sending emails, we also write a function to send emails with three parameters: the split couplelist, the email address responsible for sending, and the security key for that email address. We global a 'sent_counter' to count how many emails are sent successfully.

Each pair in the couplelist needs to send two emails: one to the first person and one to the second person. If an email is sent successfully, the program prints 'successfully sent', and 'sent_counter' increases by 1. If not, the program prints which email address failed to send, for future reference.