Code explanation

```c
struct Account {
    char *name;
    char *password;
    struct transaction transactions[MAX_TRANSACTIONS];
    unsigned int transaction_count;
    struct entity entitites[MAX_TRANSACTIONS];
    unsigned int entity_number;
};
```

The data for users is saved in an array of structs of type account which contains structs for entities and transactions, each account can hold up to 100 entities and transactions, and the data in them is allocated dynamically to make good use of memory.

```c
#define MAX_TRANSACTIONS 100
struct entity {
  char *name;
  char *contact_info;
  char *desc;
};
```

```c
struct transaction
{
    /*
    struct to store financial data
    type date and description of transaction are saved as char arrays whilst amount
    is a float value used to store the money amount
    */
    char *type, *date, *desc;
    float amount;
};
```

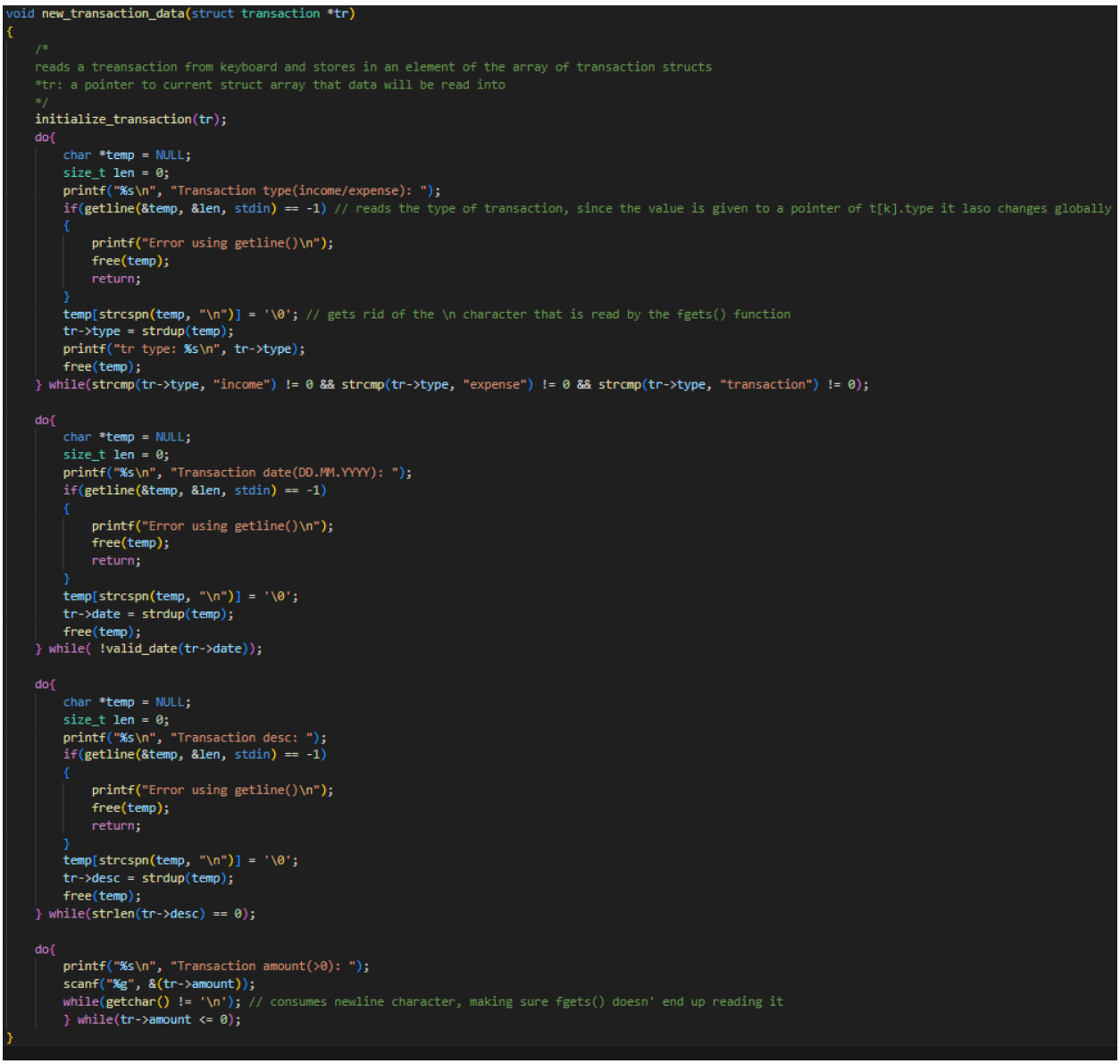
The program prints a menu which allows the user to add a new account or log into a new one. Logging into an account will call the following login function, it requires a username aswell as a password both read from the keyboard, count is the count of all current accounts the program holds so we can go through them and check if the user data is correct. If the data is correct the program returns the index of said user in order to have it used for performing needed operations, otherwise it returns -1 in case of failure.

```c
int login(char username[], char password[], unsigned count)
{
  /*
  function for user login: checks if the given userame and password exists, if they dont it returns -1
  username: char array with given username
  password: char array with given password
  count: number of acccounts
  */
  char *temp = malloc(MAX * sizeof(char)), file_name[50], file_password[50], *p;
  int k = 0;
  char path[50];
  for(unsigned k=0; k < count; k++)
  {
    if (sprintf(path, "%s%u/%u%s", "./Users/", k, k, ".txt") < 0)
    {
      printf("Error using sprintf\n");
      free(temp);
      return -1;
    }
    FILE *f = fopen(path, "r");
    if (f == NULL)
    {
      printf("Error opening file\n");
      free(temp);
      return -1;
    }

      if(fgets(temp, MAX, f) != NULL)
      {
        p = strtok(temp, " \n");
        if(p != NULL)
          {
            strcpy(file_name, p);
            p = strtok(NULL, " \n");
            if(p != NULL)
              {
                strcpy(file_password, p);
                if(strcmp(username, file_name) == 0 || strcmp(password, file_password) == 0)
                  {
                    fclose(f);
                    free(temp);
                    return k;
                  }
              }
          }
      }
    fclose(f);
  }
  free(temp);
  return -1;
}
```

If login is successful the user can manage their account, add a new transaction or entity to the account, get a financial report for a timeframe or transfer money to a different account:

The following function adds data to a transaction element and checks that it is correct, if it is not it requests new data from the user using the do…while loop:



```c
void new_transaction_data(struct transaction *tr)
{
    /*
    reads a treansaction from keyboard and stores in an element of the array of transaction structs
    *tr: a pointer to current struct array that data will be read into
    */
    initialize_transaction(tr);
    do{
        char *temp = NULL;
        size_t len = 0;
        printf("%s\n", "Transaction type(income/expense): ");
        if(getline(&temp, &len, stdin) == -1) // reads the type of transaction, since the value is given to a pointer of t[k].type it laso changes globally
        {
            printf("Error using getline()\n");
            free(temp);
            return;
        }
        temp[strcspn(temp, "\n")] = '\0'; // gets rid of the \n character that is read by the fgets() function
        tr->type = strdup(temp);
        printf("tr type: %s\n", tr->type);
        free(temp);
    } while(strcmp(tr->type, "income") != 0 && strcmp(tr->type, "expense") != 0 && strcmp(tr->type, "transaction") != 0);

    do{
        char *temp = NULL;
        size_t len = 0;
        printf("%s\n", "Transaction date(DD.MM.YYYY): ");
        if(getline(&temp, &len, stdin) == -1)
        {
            printf("Error using getline()\n");
            free(temp);
            return;
        }
        temp[strcspn(temp, "\n")] = '\0';
        tr->date = strdup(temp);
        free(temp);
    } while( !valid_date(tr->date));

    do{
        char *temp = NULL;
        size_t len = 0;
        printf("%s\n", "Transaction desc: ");
        if(getline(&temp, &len, stdin) == -1)
        {
            printf("Error using getline()\n");
            free(temp);
            return;
        }
        temp[strcspn(temp, "\n")] = '\0';
        tr->desc = strdup(temp);
        free(temp);
    } while(strlen(tr->desc) == 0);

    do{
        printf("%s\n", "Transaction amount(>0): ");
        scanf("%g", &(tr->amount));
        while(getchar() != '\n'); // consumes newline character, making sure fgets() doesn' end up reading it
        } while(tr->amount <= 0);
}
```

In order to provide a report for a timeframe we use a for loop to iterate through all transactions meeting the required dates, the dates are flipped using the flip_date() function so that the dates can be compared using strcmp():

```c
void income_breakdown_for_timeframe(struct transaction t[], int n, char starting_date[], char end_date[])
{
    /*
    calculates and prints total account balance of transactions in given timeframe currently in the struct array
    t[]: struct array that needs to have its balance calculated
    n: int value representing total lenght of array
    starting_date[]: start date of the timeframe
    end_date[]: end date of timeframe
    */
    float spent = 0, got = 0;
    char flipped_start[12], flipped_end[12];
    flip_date(starting_date, flipped_start);
    flip_date(end_date, flipped_end);

    for (int i=0; i<n; i++)
    {
        printf("%d\n", i);
    char flipped_date[12];
    flip_date(t[i].date, flipped_date);
        if(strcmp(flipped_date, flipped_start) >= 0 && strcmp(flipped_date, flipped_end) <= 0)
            if(strstr(t[i].type, "expense") != NULL)
                spent += t[i].amount;
            else
                got += t[i].amount;
    }

    if(spent !=n || got !=0)
        printf("Income for timeframe: %g \nExpenses for timeframe: %g\n", got, spent);
    else
        printf("%s", "No avalabile data for given timeframe\n");
}
```

```c
#ifdef _WIN32
#include <direct.h>
#include <Windows.h>
#define PLATFORM_MKDIR(dir) _mkdir(dir)
#define PLATFORM_RMDIR(dir) _rmdir(dir)
#else
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define PLATFORM_MKDIR(dir) mkdir(dir, 0777)
#define PLATFORM_RMDIR(dir) rmdir(dir)
#endif
```

For functions managing directories and files a macro was used to make sure the correct mkdir functions the following macro was used.

To delete an account we go to the needed directory and file and delete them then rename the remaining correctly:

```c
void delete_account(unsigned *count, unsigned index)
{
  /*
  function to delete an account and rename the folders of the remaining accounts accordingly
  count: how many accounts there are in total, prevents calling the function with an invalid value
  index: index of account to delete
  */
  char path[50],prev_name[50],new_name[50],prev_file_name[50],new_file_name[50];
  if(index >= *count)
    {
      printf("No account at given index\n");
      return;
    }

  if(sprintf(prev_name, "%s%u", "./Users/", index) < 0)
    {
      printf("sprintf error\n");
      return;
    }

  if(sprintf(prev_file_name, "%s%u/%u%s", "./Users/", index, index, ".txt") < 0)
    {
      printf("sprintf error\n");
      return;
    }

  if (remove(prev_file_name) != 0)
    {
        perror("Error removing file");
        return;
    }

  if(sprintf(prev_file_name, "%s%u/%s", "./Users/", index, "entities.txt") < 0)
    {
      printf("sprintf error\n");
      return;
    }

  if (remove(prev_file_name) != 0)
    {
        perror("Error removing file");
        return;
    }

  if (PLATFORM_RMDIR(prev_name) != 0)
    {
        perror("Error removing user directory");
        return;
    }
    index++;

  while(index < *count)
        {
            if(sprintf(prev_file_name, "%s%u/%u%s", "./Users/", index, index, ".txt") < 0)
            {
                printf("sprintf error\n");
                return;
            }

            if(sprintf(prev_name, "%s%u", "./Users/", index) < 0)
            {
                printf("sprintf error\n");
                return;
            }

            if(sprintf(new_file_name, "%s%u/%u%s", "./Users/", index, index-1, ".txt") < 0)
            {
                printf("sprintf error\n");
                return;
            }
```

```
  while(index < *count)
        {
            if(sprintf(prev_file_name, "%s%u/%u%s", "./Users/", index, index, ".txt") < 0)
            {
                printf("sprintf error\n");
                return;
            }

            if(sprintf(prev_name, "%s%u", "./Users/", index) < 0)
            {
                printf("sprintf error\n");
                return;
            }

            if(sprintf(new_file_name, "%s%u/%u%s", "./Users/", index, index-1, ".txt") < 0)
            {
                printf("sprintf error\n");
                return;
            }

            if(sprintf(new_name, "%s%u", "./Users/", index-1) < 0)
            {
                printf("sprintf error\n");
                return;
            }

            if (rename(prev_file_name, new_file_name) != 0)
            {
                perror("Error renaming file");
                return;

            }

            if (rename(prev_name, new_name) != 0)
            {
                perror("Error renaming directory");
                return;

            }
            index++;
        }
  (*count)--;
  char msg[MAX];
  if(sprintf(msg, "%s %u%s\n", "Deleted account number", (*count) + 1, ",all remaining account folder names have been updated accordingly") < 0)
    {
      printf("error using sprintf\n");
      return;
    }
  log_msg(msg);
}
```