CompanyTextFile

int saveCompanyToFile(const Company* pManager, const char* fileName);
void writeClientsArrayCompanyToFile(FILE* fp, const Client* pClient, int clientsNum);
void writeContactToFile(const Contact* pContact, FILE* fp);
void writeClientToFile(const Client* pClient, FILE* fp);
void writeCandidatesArrayCompanyToFile(FILE* fp, const Candidate** pCandidate, int CandidatesNum);
void writeCandidateToFile(const Candidate * pCandidate, FILE* fp);
void writeTeamMembersArrayCompanyToFile(FILE* fp, const TeamMember** pTeamMember, int TeamMembersNum);
void writeTeamMemberToFile(const TeamMember* pTeamMember, FILE* fp);
void writeProjectManagementArrayCompanyToFile(FILE* fp, const ProjectManagement** ProjectManagement, int ProjectManagementNum);
void writeDateToFile(const Date* pDate ,FILE* fp);
void writeProjectToFile(const Project* pProject, FILE* fp);
void writePMTeamMembersIDArrayToFile(const ProjectManagement* pProjectManagement, FILE* fp);
void writeTaskToFile(const Task* pTask, FILE* fp);
void writeProjectManagementToFile(const ProjectManagement* pProjectManagement, FILE* fp);
void writeClientWebsiteRequirements(const Project* pProject, FILE* fp);
void writePMTasksToFile(const ProjectManagement * pProjectManagement, FILE* fp);
void writePaymentsToFile(const Payment* pPayment, FILE* fp);

int readCompanyFromFile(Company* pCompany, const char* fileName);
int readContactFromFile(Contact* pContact, FILE* fp);
int readClientFromFile(Client* client, FILE* fp);
int readClientsArrayFromFile(Company* pCompany, FILE* fp);
int readCandidatesArrayCompanyFromFile(Company* pCompany, FILE* fp);
int readCandidateCompanyFromFile(Candidate * pCandidate, FILE* fp);
int readTeamMemberCompanyFromFile(TeamMember* pTeamMember, FILE* fp);
int readTeamMembersArrayCompanyFromFile(Company* pCompany, FILE* fp);
int readProjectManagementsArrayCompanyFromFile(Company* pCompany, FILE* fp);
int readProjectManagementFromFile(ProjectManagement* pProjectManagement, TeamMember** teamMemberArr,int amount, FILE* fp);
int readWebsiteReq(Project * pProject, FILE* fp);
int readProjectFromFile(Project * pProject, FILE* fp);
int readDateFromTextFile(Date * pDate, FILE* fp);
int readFeedbackFromTextFile(Feedback * pFeedback, FILE* fp);
int readTeamMembersIDFromFile(ProjectManagement* pProjectManagement, TeamMember** teamMemberArr, int amount, FILE* fp);
int readPMTasksFromFile(ProjectManagement* pProjectManagement, FILE* fp);
int readPMTaskFromFile(Task* pTask, ProjectManagement* pProjectManagement, FILE* fp);
int readPaymentFromFile(Payment* pPayment, FILE* fp);

//HELPER
void removeNewLine(char *str);
int addClientFromFile(Company* pCompany , Client* client , int num);
void freeClientArrayFileVersion(Company* pCompany,int num);
void freeCandidatesArrayFileVersion(Company* pCompany,int);
int addCandidateFromFile(Company* pCompany , Candidate * candidate , int num);
void freeTeamMembersArrayCompanyFileVersion(Company* pCompany,int num);
int addTeamMemberFromFile(Company* pCompany , TeamMember* pTeamMember , int num);
void freeTeamMembersArrayPMFileVersion(ProjectManagement* pProjectManagement,int num);
void freeProjectManagementArrayFileVersion(Company* pCompany, int num);

CompanyBinaryFile

int saveCompanyToBinaryFile(const Company* pCompany, const char* fileName);
void writeContactToBinaryFile(const Contact* pContact, FILE* fp);
int writeClientsArrayToBinaryFile(const Company* pCompany, FILE* fp);
void writeClientToBinaryFile(const Client* pClient, FILE* fp);
int writeCandidatesArrayToBinaryFile(const Company* pCompany, FILE* fp);
void writeCandidateToBinaryFile(const Candidate * pCandidate, FILE* fp);
int writeTeamMembersArrayToBinaryFile(const Company* pCompany, FILE* fp);
void writeTeamMemberToBinaryFile(const TeamMember * pTeamMember, FILE* fp);
int writeProjectManagementsArrayToBinaryFile(const Company* pCompany, FILE* fp);
void writeProjectManagementToBinaryFile(const ProjectManagement * pProjectManagement, FILE* fp);
void writeProjectToBinaryFile(const Project * pProject, FILE* fp);

int readCompanyFromBinaryFile(Company* pCompany, const char* fileName);
int readContactFromBinaryFile(Contact* pContact, FILE* fp);
int readClientsArrayFromBinaryFile(Company* pCompany, FILE* fp);
int readClientFromBinaryFile(Client* pClient, FILE* fp);
int readCandidatesArrayFromBinaryFile(Company* pCompany, FILE* fp);
int readCandidateFromBinaryFile(Candidate * pCandidate, FILE* fp);
int readTeamMembersArrayFromBinaryFile(Company* pCompany, FILE* fp);
int readTeamMemberFromBinaryFile(TeamMember* pTeamMember, FILE* fp);
int readProjectManagementsArrayFromBinaryFile(Company* pCompany, FILE* fp);
int readProjectManagementFromBinaryFile(ProjectManagement* pProjectManagement, TeamMember** teamMemberArr, int amount, FILE* fp);
int readProjectFromBinaryFile(Project * pProject, FILE* fp);

//BinaryFile Helper
void writeDateToBinaryFile(const Date *date, FILE *fp);
void writeStringToBinaryFile(FILE* file, const char* string);
void writeWebsiteRequirements(char **websiteReq, int reqNumber,FILE *file);
void writePMTeamMembersIDArrayToBinaryFile(const ProjectManagement* pProjectManagement, FILE* fp);
void writePMTaskToBinaryFile(const Task* pTask, FILE* fp);
void writePMTasksToBinaryFile(const ProjectManagement* pProjectManagement, FILE* fp);
void writePaymentToBinaryFile(const Payment *pPayment, FILE *fp);

//BinaryRead Helper
char* readStringFromFile(FILE *file);
int readWebsiteReqBinary(Project *pProject, FILE *fp);
int readPMTeamMembersIDArrayFromBinaryFile(ProjectManagement* pProjectManagement, TeamMember** teamMemberArr, int amount, FILE* fp);
int readTasksArrayFromBinaryFile(ProjectManagement* pProjectManagement, FILE* fp);
int readyTaskFromBinaryFile(ProjectManagement* pProjectManagement,Task* pTask,FILE* fp);
int readPaymentFromBinaryFile(Payment * pPayment, FILE* fp);

MACROS

#define CHECK_RETURN_INT(*ptr, str*);
#define MINUS(*x,y*);
#define CHOOSE(*num,x,y*);
#define ZERO_COUNTER(*ptr,str*);
#define PRINT_SEPARATION();
#define BREAK_IF_0(*num,str*)

Company Struct

typedef *enum* {
eID, eName,eSpecialization, eNotSorted,eNofOption
} eSort;
static const char* enumSort

Contact info;
ProjectManagement** projectManagementsArray;
TeamMember** teamMembersArray;
Client* clientsArray;
Candidate** candidatesArray;
eSort sort;
int numOfProjects;
int numOfClients;
int numOfTeamMembers;
int numOfCandidates;
int companyEarning;

/Functions
int initCompany(Company* pCompany);
int initCompanyFromFile();
void printCompany(const Company* pCompany);
int addProjectManagement(Company* pCompany);
int removeProjectManagement(Company* pCompany);
void removeProjectManagementHelper(Company* pCompany, int num);
int addTeamMemberToArray(Company* pCompany , Candidate* pCandidate);
void removeTeamMembersFromProjectManagement(ProjectManagement* pProjectManagement);
int removeTeamMemberFromArray(Company* pCompany);
void removeTeamMemberFromArrayInAllProjectManagement(Company* pCompany, TeamMember* pTeamMember);
void findTeamMemberPlacementAndDelete(ProjectManagement * pProjectManagement, TeamMember* pTeamMember);
int addCandidate(Company* pCompany);
int removeCandidate(Company* pCompany);
int candidateApproval(const Candidate* pCandidate);
int addClientToArray(Company* pCompany);
void freeCompany(Company* pCompany);
void freeTeamMembersArray(Company* pCompany);
void freeCandidatesArray(Company* pCompany);
void freeProjectManagementsArray(Company* pCompany);
void freeClients(Company* pCompany);
TeamMember* findTeamMemberWithTheSameIdInCompany(TeamMember ** TeamMemberArray,int amount, int numIdTeamMember);

//Helper Functions
Client* findClient(Company* pCompany);
void sortClient(Company* pCompany);
eSort getSort();
int deleteTeamMemberFromArrayInCompany(Company* pCompany, int num);
int deleteCandidateInArray(Company* pCompany, int num);
void addObjectByUserPreference(Company* pCompany);
void removeObjectByUserPreference(Company* pCompany);
Candidate* pickCandidate(const Company* pCompany ,int* choose);
ProjectManagement* pickProjectManagement(const Company* pCompany);
TeamMember* pickTeamMember(const Company* pCompany);
void PrintObjectByUserPreference(Company* pCompany);
int payProjectPayments(Company* pCompany);
int addProjectManagementFromFile(Company* pCompany , ProjectManagement * projectManagement , int num);

//Menu
int menuAddCompany();
int menuRemoveCompany();
int menuPrintCompany();

ProjectManagment Struct

Project project;
int numberOfTeamMembers;
TeamMember** projectTeamMembersArray;
int numberOfTasks;
LIST projectManagementTasksLists;
Payment payment;

//Functions
int initProjectManagement(ProjectManagement* PM,Client* client);
void printProjectManagement(const void* projectManagement);
void printProjectManagementShort(const void* projectManagement);
int addTeamMemberToProjectManagement(ProjectManagement* PM , TeamMember* TM);
int removeTeamMemberFromProjectManagement(ProjectManagement* PM);
int removeTeamMemberFromProjectManagementHelper(ProjectManagement* PM, int num);
int removeAllTasksOfTheProjectManagementTeamMemberList(ProjectManagement* PM, NODE * headListTeamMemberTask);
int addTask(ProjectManagement* PM);
int addTaskToProManList(ProjectManagement* PM, Task* task);
void createCopyTask(Task* taskCopy,const Task* task);
int deleteTask(ProjectManagement* PM);
int chooseTask(ProjectManagement* PM);
void projectManagementMenu(ProjectManagement* PM);
int projectManagementMenuHelper();
void printTeamMemberTasksYouChoose(const ProjectManagement* PM);
void freeProjectManagement(ProjectManagement* PM);

//Helper Functions
int deleteTeamMemberInArray(ProjectManagement* PM, int num);
int printProjectTeamMembersArray(const ProjectManagement* PM);
TeamMember* getTeamMember(TeamMember** teamMembersArray,int);
TeamMember* getTeamMemberByType(TeamMember** teamMembersArray,int,int);
void printPMTasksList(const ProjectManagement* PM);
int removeAllTasksOfTheProjectManagement(ProjectManagement* PM, TeamMember* teamMember);
int removeTaskOfTheProjectManagement(ProjectManagement* PM, int skip);
void findTeamMemberWithTheSameIdAndDeleteTask(ProjectManagement* PM, int numIdTeamMember, int numIdTask);
TeamMember* findTeamMemberWithTheSameId(const ProjectManagement* PM, int numIdTeamMember);
int changeTaskStatusProjectManagement(ProjectManagement* PM);
int switchLists(TeamMember* TM,NODE* pTemp, NODE* headListFrom, NODE* headListTo);
int changeTaskStatusProjectManagementHelper(ProjectManagement* PM, int statusBefore, int newStatus, NODE* pNode);
int changeProjectStatus(ProjectManagement* PM);

def (FILE NOT STRUCT)

DONE

typedef void* *DATA*;
typedef enum {False, True} *BOOL*;
define **EMPTY** 0x7FFFFFFF----------check it----------
NODE node;----------check it----------

typedef struct node{ *DATA* key; struct node* next; }*NODE*;

Candidate Struct

int experience;
eSpecialization speciality;
Contact info;

//Functions
int initCandidate(Candidate* pCandidate);
void printCandidate(const void* candidate);
void freeCandidate(Candidate* pCandidate);

//Helper Functions
void welcomeCandidate();

LIST Struct

Node head;

BOOL L_init(LIST* pList);
NODE* L_insert(NODE* pNode, DATA Value);
BOOL L_delete(NODE* pNode , void (*freeKey)(void*));
NODE* L_find(NODE* pNode, DATA value, int(*compare)(const void*, const void*));
BOOL L_free(LIST* pList,void (*freeKey)(void*));
int L_print(const LIST* pList , void (*print)(const void*));
int L_printWithNumbers(const LIST* pList , void (*print)(const void*));
int L_count(const LIST* pList);

General Struct

typedef enum {*eFrontendDeveloper*, *eBackendDeveloper*, *eDesigner*, *eProjectManager*, *eCEO*, *eNULL*, *eNotSorted*, *eNofOption*
} eSpecialization;

static const char* enumSpeciality[*eNofOption*]
= { "Front-end Developer", "Back-end Developer",
"Designer" ,"ProjectManager","CEO","NULL","Not Sorted" };

//----------
typedef enum {*eWaiting*, *eWorking*, *eDone*, *eStatusNofOption*} eStatus;

static const char* enumStatus[*eStatusNofOption*]
= { "Waiting", "Working on It","Done"};

//General Functions
char* getStrExactName(const char* msg);
char** splitCharsToWords(char* str, int* pCount, int* pTotalLength);
void generalArrayFunctionWithNumbers(void* arr, int size, size_t typeSize , void (*f)(const void* element));

//Helper Functions
char* getDynStr(char* str);
char* myGets(char* buffer, int size);
eSpecialization getSpeciality();
void printSpeciality();
int getExperience();

TeamMember Struct

static int teamMemberID;
int id;
int experience;
Contact info;
eSpecialization speciality;
int numberOfTasks;
LIST openTasksList;
LIST closedTasksList

//Functions
int initTeamMember(TeamMember* pTeamMember);
void addToTeamMemberID(int num);
void printTeamMember(const void* teamMember);
void printTeamMemberTasks(const void* teamMember);
int insertByID(TeamMember* TM, Task* pTask, NODE* node); // Forward declaration of struct Task
int deleteTaskOfTeamMemberByID(TeamMember* TM, int numIdTask);
void freeTeamMember(TeamMember* TP1);
int insertByID_TASKS(NODE* pNode, Task* pAir);
NODE* changeTaskStatusOfTeamMemberByID(TeamMember* TM, int numIdTask, int status);
NODE* changeTaskStatusOfTeamMemberByIDHelper(NODE* node, int id, int status);
int turnToTeamMember(TeamMember* pTeamMember , Candidate* pCandidate);
int sameTeamMemberID(const void* TM1,const void* TM2);

//Helper Functions
int isTheSameTeamMemberId(TeamMember* TP1, int TP2);
int switchDepartment(TeamMember* TP1);
int deleteTaskOfTeamMemberByIDHelper(NODE* headList, int id);

Project Struct

char* name;
bool zoomMeeting;
char** websiteReq;
int reqNumber;
Client client;
Date dueDate;
eStatus projectStatus;
Feedback* feedback;

//Functions
int initProject(Project* pProject, Client* client);
void printProject(const void* project);
int getClientWebsiteRequirements(Project* pProject);
int writeWebsiteReq(Project* pProject);
void isZoomMeeting(Project* pProject);
void printWebsiteRequirements(const Project* pProject);
void freeProject(Project* pProject);

//Helper Functions
int addFeedback(Project* pProject);
void freeWebsiteReqArr(char** arr, int size);
void zoomMeetingOccurred(Project* pProject);

Client Struct

static int clientID
;
int Id;
Contact info;

int initClient(Client* pClient);
void addToClientID(int num);
void printClient(const void* client);
int compareClientsByID(const void* C1,const void* C2);
int compareClientsByName(const void* C1,const void* C2);
int compareClientsByPhoneNumber(const void* C1,const void* C2);
void freeClient(Client* pClient);

Payment Struct

static int paymentID;
typedef enum {*ePending,eCompleted,eNotDecided,eNumofOption*} ePayment;
static const char* enumPayment[*eNumofOption*] = {"Pending", "Completed", "Waiting"};

int tasksNumber;
int total;
int id;
int NofPayments;
Date paymentDate;
ePayment status;

int initPayment(Payment* pPayment, int numOfTasks);
void printPayment(const Payment* pPayment);
int payPayment(Payment* p1);
int payPaymentHelper(Payment* p1);
void addTasksToTasksNumberPayment(Payment* p1, int plus);
void deleteTasksToTasksNumberPayment(Payment* p1, int less);

Contact Struct

char* name;
char phoneNumber[LengthOfPhoneNum + 1];
char* email;

//Functions
int initContact(Contact* pContact, char* str);
int getContactName(Contact* pContact, char* str);
char* allocateRightLength(char** wordsArray,int count, int totalLength);
void changeName(char* name, int count, char** wordsArray);
void getPhoneNumber(char* phoneNumber);
int getEmail(Contact* pContact);
int isValidEmail(const char *email);

//Helper Function
void printContact(const Contact* pContact);
int isTheSameContactPhone(Contact* pContact1,Contact* pContact2);
int isTheSameContactEmail(Contact* pContact1,Contact* pContact2);
void freeContact(Contact* pContact);

Feedback Struct

char* feedbackStr;
int rate

//Functions
int initFeedback(Feedback* pFeedback , Client* client);
void printFeedback(const Feedback* pFeedback);
int editFeedback(Feedback* pFeedback);
void freeFeedback(Feedback* pFeedback);

//Helper Functions
void getRate(Feedback* pFeedback);
char* getFeedbackString(Feedback* pFeedback);

Date Struct

unsigned int year;
unsigned int month;
unsigned int day;

int compareDate(Date *d1*,Date *d2*);

void getCorrectDate(Date* pDate);
int checkDate(char* date, Date* pDate);
void printDate(const Date* pDate);

int saveDateToFile(FILE* fp,const Date* pDate);
int readDateFromBinaryFile(Date* pDate, FILE* fp);

Task Struct

static int taskID = 1;
int id;
char* taskStr;
struct TeamMember* teamMember;
eStatus taskStatus;

//Functions
int initTask(Task* pTask, struct TeamMember* teamMember);
void addToTaskID(int num);
void printTask(const void* pTask);
void freeTask(void* pTask);
int changeTaskStatus(Task* t1);

//Helper Functions
int changeStatus(Task* t1, int option);
int compareTaskByID(const void* T1, const void* T2);
int sameTaskID(const void* T1, const void* T2);
int returnTeamMemberID(const void* T1);
int returnIdTask(const void* T1);
int getTaskStatus(const void* T1);
int sameTaskIDByInt(const void* T1,const void* id);