מערכת לניהול בנק – דו"ח פרויקט

מגישים:

קריסטינה גולדין 317958700

דוד בן יעקב 320759921

הקדמה

בפרויקט זה בנינו מערכת שמנהלת בנק, את כל סניפיו עובדיו ולקוחותיו בשפת C בסביבת Visual Studio.

המערכת מנטרת את כל הסניפים של הבנק, שומרת את כל הפרטים של הסניפים ומאפשרת להוסיף או למחוק סניפים מהרשת.

לכל סניף יש את העובדים שלו, המערכת שומרת את כל הפרטים של עובדי הבנק, ומאפשרת לנהל את העובדים ואת המשכורות שלהם, בנוסף מאפשרת הוספה ומחיקה של עובדים מהרשת.

יתר על כן, המערכת שומרת את כל המידע על לקוחות הבנק, פרטיהם האישיים ויוצרת חשבון בנק ייחודי לכל לקוח של הבנק. כל חשבון בנק כולל את מספר החשבון ואת הסניף אליו מקושר החשבון.

המערכת גם מאפשרת ניהול, הוספה ומחיקה של לקוחות בבנק וניטור שוטף של כל הלקוחות.

מבני המערכת (Structs)

מבנים בונים:

1. כתובת:

```
#define SIZE 20

typedef struct
{
	char country[SIZE];
	char city[SIZE];
	char street[SIZE];
	int number;
}Address;
```

מבנה הכתובת מכיל ארץ, עיר, רחוב (מחרוזות בגודל קבוע של 20 תווים) ומספר מבנה (integer).

2. תאריך:

```
typedef struct
{
	int day;
	int month;
	int year;
}Date;
```

מבנה התאריך מכיל יום, חודש ושנה (integer).

3. רשימה מקושרת חד כיוונית:

```
typedef struct node1 {
	void* data;
	struct node1* next;
}NODE1;

typedef struct {
	NODE1 head;
	int data_size;
}LIST1;
```

המבנה NODE1 (צומת) מאפיין איבר ברשימה ומכיל מצביע מסוג void, מה שהופך את הרשימה במקושרת לרשימה גנרית, כלומר כל סוג של מידע יכול להישמר ברשימה זאת בעזרת מצביעים.
בנוסף, המבנה מכיל מצביע לערך הבא ברשימה מסוג node.

המבנה LIST1 מכיל את הכותרת head של הרשימה שהוא מסוג NODE1 ומכיל את גודל המידע שהולכת להכיל הרשימה, בפרויקט שלנו הרשימה תכיל מידע מסוג Branch אשר נפרט עליו בהמשך.
הגודל מיוצג על ידי משתנה מסוג integer.

4. רשימה מקושרת דו כיוונית:

```
typedef struct node2 {
	Customer data;
	struct node2* next;
	struct node2* prev;
}NODE2;
```

המבנה NODE2 מאפיין איבר ברשימה המקושרת הדו כיוונית.
המבנה מכיל מידע מסוג Customer עליו נפרט בהמשך, ושני מצביעים, אחד לאיבר הבא ברשימה ואחד לאיבר הקודם ברשימה מסוג node2.

מבנים מרכזיים:

1. בנק:

```
typedef struct
{
	int bank_code;
	char* name;
	LIST1 branchList;
	int numOfBranches;
}Bank;
```

מבנה הבנק מכיל את קוד הבנק (integer), שם הבנק (מצביע למחרוזת/מחרוזת דינאמית), רשימת כל סניפי הבנק שממשומשת בעזרת רשימה מקושרת חד-כיוונית (כותרת הרשימה LIST1) ומספר הסניפים בבנק (integer).

2. פרטים אנושיים:

```
#define SIZE 20

typedef struct
{
	int id;
	char name[SIZE];
	Date dateOfBirth;
	Address address;
}HumanData;
```

מבנה זה מכיל את כל המידע האנושי הבסיסי, מבנה זה יוכל במבנים של העובדים והלקוחות על מנת למנוע כפילות בקוד.

המבנה מכיל מספר תעודת זהות (integer), שם (מחרוזת בגדול קבוע של 20 תווים), תאריך (מבנה מסוג Date) וכתובת מגורים (מבנה מסוג Address).

3. עובד:

```
typedef enum { employee, manager, teamLeader, ShiftManager, NumRoles }
employee_role;

typedef struct
{
	HumanData emploData;
	double salary;
	employee_role role;
}Employee;
```

מבנה עובד מכיל מידע על העובד (מבנה מסוג HumanData), משכורת העובד (double) ותפקיד העובד בבנק אשר מיוצג על ידי enum כפי שמתואר מעלה.

4. סניף:

```
typedef struct
{
	int code;
	char* name;
	Address address;
	Employee** employees;
	int numOfEmployees;
}Branch;
```

מבנה הסניף מכיל את קוד הסניף (integer), שם הסניף (מצביע למחרוזת/מחרוזת דינאמית), כתובת הסניף (מבנה Address), רשימה של עובדי הבנק (מערך דינאמי של מצביעים למבנים מסוג Employee) ומספר העובדים בסניף (integer).

5. חשבון בנק:

```
typedef struct
{
	int number;
	Branch* branch;
}BankAccount;
```

מבנה זה מכיל מספר חשבון (integer) ומצביע למבנה Branch אשר מייצג את הסניף שמקושר לחשבון הבנק.

6. לקוח:

```
typedef struct
{
	HumanData customerData;
	Date dateOfJoin;
	BankAccount* account;
}Customer;
```

מבנה לקוח מכיל מידע בסיסי על הלקוח (מבנה מסוג HumanData), תאריך הצטרפות לבנק (Date) וחשבון הבנק המקושר ללקוח (מצביע למבנה מסוג BankAccount).

7. ניהול עובדים:

```
typedef enum { notSorted, byName, byRole, bySalary, NumOfSort } sort_type;

typedef struct
{
	Employee* employees;
	int numOfEmployees;
	sort_type sortBy;
}EmployeeManager;
```

מבנה ניהול עובדים מכיל את רשימת העובדים הכללית של הבנק (מערך דינאמי המכיל מבנים מסוג Employee), מספר העובדים ברשת (integer), ושיטת מיון רשימת העובדים אשר מיוצגת בenum המפורט מעלה.

8. ניהול לקוחות:

```
typedef struct
{
	NODE2 customerList;
	int numOfCustomers;
}CustomerManager;
```

מבנה ניהול לקוחות מכיל רשימה של כל לקוחות הבנק. הרשימה מממשת את המבנה NODE2 אשר מאפיין איבר ברשימה מקושרת דו כיוונית. בנוסף, המבנה מכיל את מספר הלקוחות הכולל בבנק (integer).

פעולות המערכת

תפריט ראשי

1. הוספת סניף בנק חדש למערכת
2. הדפסת כל פרטי הבנק ואת כל סניפיו
3. הוספת לקוח חדש למערכת
4. הדפסת כל לקוחות הבנק
5. הוספת עובד חדש למערכת
6. הדפסת כל עובדי הבנק
7. מיון עובדים
8. חיפוש עובדים
9. עדכון משכורת של עובד
10. הדפסת כל חשבונות הבנק
11. הדפסת כל העובדים של סניף מסוים
12. מחיקת ערכים מהבנק (סניפים/עובדים/לקוחות)
13. יציאה מהתוכנית

קבצים

בכדי לעלות תכנית מקובץ ולדעת לבצע שינויים בקבצים, בין אם הם בינאריים ובין אם הם קבצי טקסט, אנחנו צריכים לדעת את הסדר כתיבה של נתונים בתוך הקובץ.

להלן מבני הקבצים במערכת:

קובץ סניפים:

[Branch Code]
[Branch Name]
[Country]
[City]
[Street]
[Number]

קובץ עובדים:

[ID]
[Name]
[Date of Birth]
[Country]
[City]
[Street]
[Number]
[Salary]
[Employee Role]
[Branch Code]

קובץ לקוחות:

[ID]
[Name]
[Date of Birth]
[Country]
[City]
[Street]
[Number]
[Date of Join]
[Bank Account Number]
[Branch Code]

קבצים בינאריים

במידה והמשתמש בוחר העלאת המידע השמור על המערכת מקבצים בינאריים, המערכת תפתח לקריאה את הקבצים הרלוונטיים בלבד.

לעומת זאת, שמירת המידע מתבצעת בשני סוגי הקבצים במקביל, המערכת מאפשרת שמירה זאת רק לאחר אתחול ראשוני של המידע במבנים הרלוונטיים.

כאשר מוסיפים מבנה כלשהו, המשתמש ממלא את הערכים הרלוונטיים למבנה שנבחר, ולאחר מכן המערכת כותבת את הנתונים הרלוונטיים בקבצים המתאימים על-ידי פתיחתם במוד "ab" – append binary.

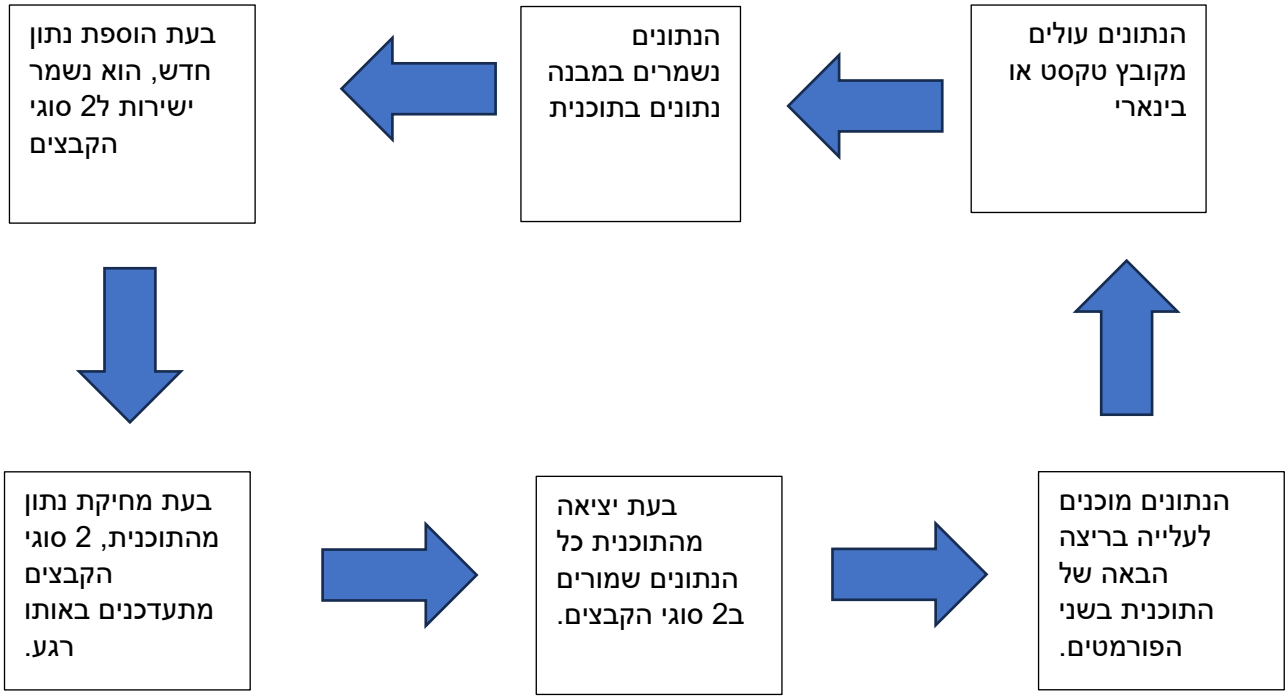
כאשר פותחים את הקובץ באופן זה, המערכת רושמת את הנתונים החדשים בסוף הקובץ הקיים, דבר המאפשר הרחבה של פעילות המערכת במידת הצורך.

המבנים אשר נשמרים במערכת באופן זה הם: Employee , Branch, Customer, אולם לא כל הנתונים המרכיבים את המבנה בקובץ הC בהכרח נשמרים בקבצים הבינאריים משום קשרים פנימיים הקיימים בין המבנים השונים.

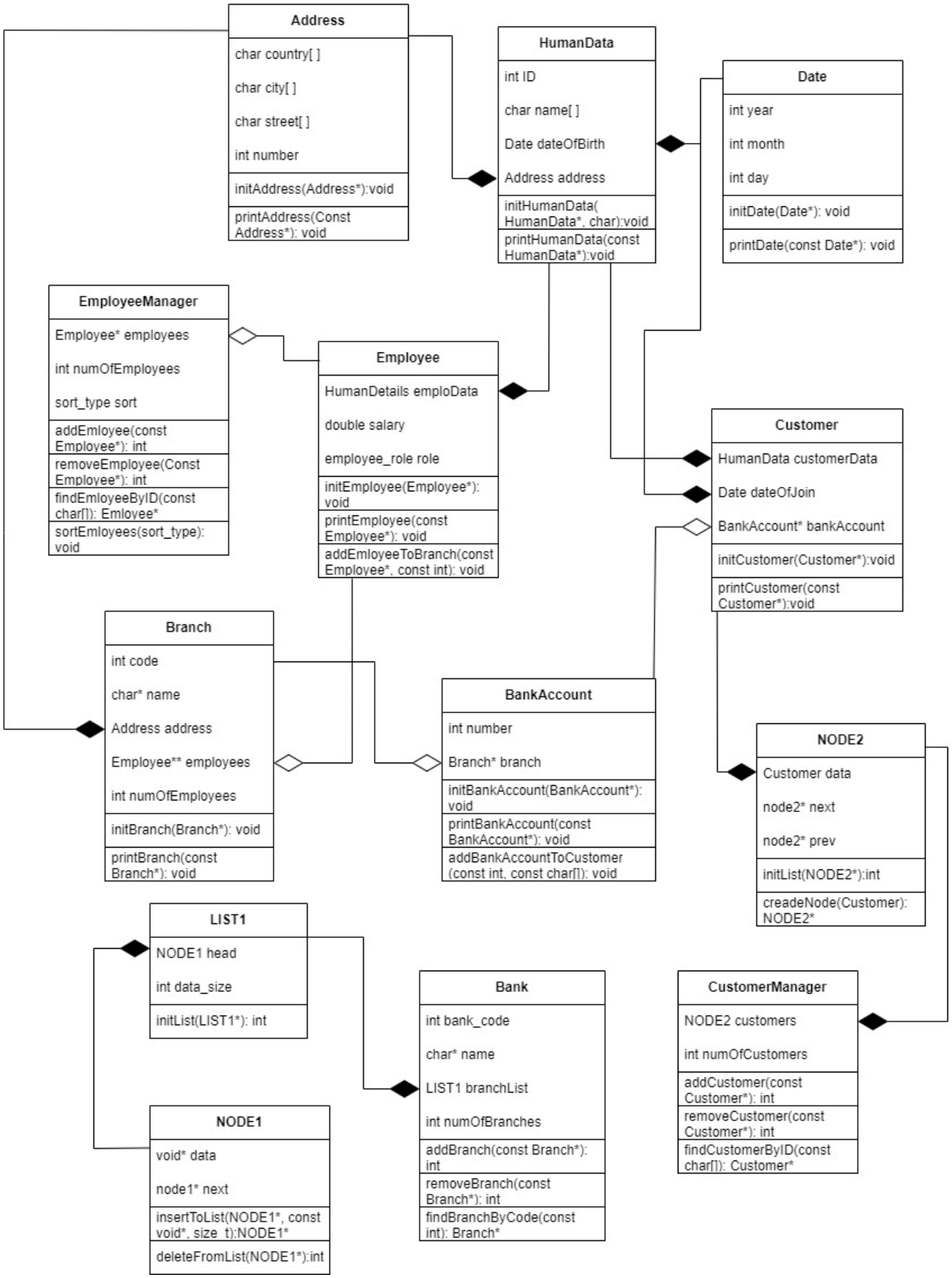
בנוסף, המערכת מאפשרת מחיקה של כל אחד מהמבנים הנ"ל, ובכדי לאפשר אמיתות הדברים בעת הפעלה מחודשת של המערכת ישנו עדכון של הקבצים בעת בקשת מחיקה של אחד הנתונים השמורים במערכת.

עדכון הנתונים מחדש מתבצע במוד "wb " – write binary.

- כל המוזכר לעיל תקף גם לגבי קבצי הטקסט אשר נשמרים אוטומטית גם כן לכל מבנה שצוין.



שרטוט מערכת


Address
char country[ ]
char city[ ]
char street[ ]
int number
initAddress(Address*):void
printAddress(Const Address*): void
HumanData
int ID
char name[ ]
Date dateOfBirth
Address address
initHumanData(HumanData*, char):void
printHumanData(const HumanData*):void
Date
int year
int month
int day
initDate(Date*): void
printDate(const Date*): void
EmployeeManager
Employee* employees
int numOfEmployees
sort_type sort
addEmloyee(const Employee*): int
removeEmployee(Const Employee*): int
findEmloyeeByID(const char[]): Emloyee*
sortEmloyees(sort_type): void
Employee
HumanDetails emploData
double salary
employee_role role
initEmployee(Employee*): void
printEmployee(const Employee*): void
addEmloyeeToBranch(const Employee*, const int): void
Customer
HumanData customerData
Date dateOfJoin
BankAccount* bankAccount
initCustomer(Customer*):void
printCustomer(const Customer*):void
Branch
int code
char* name
Address address
Employee** employees
int numOfEmployees
initBranch(Branch*): void
printBranch(const Branch*): void
BankAccount
int number
Branch* branch
initBankAccount(BankAccount*): void
printBankAccount(const BankAccount*): void
addBankAccountToCustomer (const int, const char[]): void
NODE2
Customer data
node2* next
node2* prev
initList(NODE2*):int
creadeNode(Customer): NODE2*
LIST1
NODE1 head
int data_size
initList(LIST1*): int
Bank
int bank_code
char* name
LIST1 branchList
int numOfBranches
addBranch(const Branch*): int
removeBranch(const Branch*): int
findBranchByCode(const int): Branch*
CustomerManager
NODE2 customers
int numOfCustomers
addCustomer(const Customer*): int
removeCustomer(const Customer*): int
findCustomerByID(const char[]): Customer*
NODE1
void* data
node1* next
insertToList(NODE1*, const void*, size_t):NODE1*
deleteFromList(NODE1*):int

פירוט פונקציות

תאריך:

```
void initAddress(Address* address);
```
הפונקציה מקבלת מצביע לכתובת ומאתחלת כתובת חדשה ע"י קליטת נתונים מהמשתמש.
```
int readAddressFromFile(FILE* f, Address* a);
```
הפונקציה מקבלת מצביע לכתובת ולקובץ וקוראת את נתוני התאריך מהקובץ.
```
void printAddressDataToFile(FILE* f, const Address* address, int fileType);
```
הפונקציה מקבלת מצביע לכתובת, קובץ וסוג קובץ ומוסיפה את נתוני התאריך לקובץ.
```
void printAddress(const void* a);
```

הפונקציה מדפיסה את התאריך למסך.

בנק:

```
void initBank(int state);
```
הפונקציה מאתחלת בנק.
```
void continueBankInit(FILE* f, Branch* b, int fileType);
```
הפונקציה מקבלת את סוג הקובץ ממנו יש לאתחל את הבנק.
```
void printBankBranches();
```
הפונקציה מדפיסה את כל סניפי הבנק למסך.
```
int addBranch(const Branch* branch, int fileType);
```
הפונקציה מקבלת סניף ומוסיפה אותו לבנק.
```
int removeBranch(const Branch* branch);
```
הפונקציה מקבלת סניף ומוחקת אותו מהרשימה ומהקבצים.
```
Branch* findBranchByCode(const int code);
```
הפונקציה מקבלת קוד ומחזירה את הסניף שמתאים לקוד הזה.
```
void printBankDetails();
```
הפונקציה מדפיסה למסך את פרטי הבנק.
```
void freeEmployeeArraysInBranches();
```
הפונקציה מבצעת שחרור הזיכרון שהוקצע למערכים של עובדי הסניפים.
```
void freeBank();
```
הפונקציה משחררת את הרשימה המקושרת ששמרה את כל סניפי הבנק.
```
Branch* findEmployeeBranch(const Employee* employee);
```
הפונקציה מקבלת מצביע לעובד ומחזירה באיזה סניף הוא עובד.

חשבון בנק:

```
void initBankAccount(BankAccount* account);
```
הפונקציה מקבלת מצביע לחשבון בנק ומאתחלת אותו.
```
void printBankAccountDataToFile(FILE* f, const BankAccount* account, int
fileType);
```
הפונקציה מכניסה את נתוני החשבון בנק לקובץ.
```
void printBankAccount(const void* account);
```
הפונקציה מדפיסה למסך את נתוני חשבון הבנק.
```
int readBankAccountFromFile(FILE* f, BankAccount* account, int fileType);
```
הפונקציה קוראת נתונים מקובץ.
```
Void addBranchToAccount(BankAccount* account);
```
הפונקציה מוסיפה סניף לחשבון בנק שהיא מקבלת.

סניף:

```
void initBranch(Branch* branch);
```
הפונקציה מקבלת מצביע לסניף ומאתחלת אותו.
```
Employee** initEmployeesArray(Branch* b);
```
הפונקציה מקבלת מצביע לסניף ומאתחלת מערך של עובדים עבורו.
```
int readBranchFromFile(FILE* f, Branch* b, int fileType);
```
הפונקציה קוראת את נתוני הסניף מהקובץ לפי סוג קובץ שהיא מקבלת.
```
void printBranchDataToFile(FILE* f, const Branch* branch, int fileType);
```
הפונקציה מדפיסה את ערכי הסניף לקובץ הנתון.
```
void printBranch(const void* branch);
```
הפונקציה מדפיסה את ערכי הסניף למסך.

```
void addEmployeeToBranch(const Employee* e, Branch* b);
```
הפונקציה מקבלת עובד וסניף ומוסיפה את העובד למערך העובדים של הסניף.
```
void deleteEmployeeFromBranch(const Employee* e, Branch* b);
```
הפונקציה מוחקת את העובד מהמערך של עובדי הסניף.
```
int isSameBranch(const Branch* b1, const Branch* b2);
```
הפונקציה מקבלת שני עובדים ומשווה ביניהם, מחזירה 1 אם הם שווים.
```
void printBranchEmployees(const Branch* b);
```
הפונקציה מדפיסה את כל העובדים של הסניף.
```
void freeEmployeesInArray(Branch* pB);
```
הפונקציה מבצעת שחרור של הזיכרון שהוקצע עבוד מערך העובדים.
```
Int findEmployeeInBranch(const Branch* branch, const Employee* employee);
```
הפונקציה מקבלת עובד וסניף ומחזירה 1 אם העובד עובד בסניף ו0 אם לא.

לקוח:

```
void initCustomer(Customer* c);
```
הפונקציה מקבלת מצביע ללקוח ומאתחלת אותו.
```
int readCustomerFromFile(FILE* f, Customer* c, int fileType);
```
הפונקציה קוראת את נתוני הלקוח מקובץ נתון על פי הסוג קובץ.
```
void printCustomer(const void* customer);
```
הפונקציה מדפיסה את נתוני הלקוח למסך.
```
void printCustomerDataToFile(FILE* f, const Customer* customer, int fileType);
```
הפונקציה מדפיסה את נתוני הלקוח לתוך הקובץ.
```
int isSameCustomer(const Customer* c1, const Customer* c2);
```
הפונקציה מקבלת שני לקוחות ומשווה ביניהם, מחזירה 1 אם הם שווים.
```
void freeBankAccountFromCustomer(void* c);
```
הפונקציה משחררת את הזיכרון שהוקצע עבור חשבון הבנק של הלקוח.

ניהול לקוחות:

```
void initCustomerManager(int fileType);
```
הפונקציה מאתחלת את רשימת הלקוחות.
```
void continueCustomerInit(FILE* f, Customer* c, int fileType);
```
הפונקציה מקבלת קובץ ולקוח ומתחילה להוסיף לקוחות המקובץ הנתון.
```
void printCustomerManager();
```
הפונקציה מדפיסה את רשימת הלקוחות למסך.
```
Void printBankAccounts();
```
הפונקציה מדפיסה את כל החשבונות בנק למסך.
```
int addCustomer(const Customer* customer, int fileType);
```
הפונקציה מוסיפה לקוח לרשימת הלקוחות.
```
int removeCustomer(const Customer* customer);
```
הפונקציה מוחקת לקוח מרשימת הלקוחות.
```
Customer* findCustomerByID(const int id);
```
הפונקציה מקבלת מספר תעודת זהות ומחזירה את הלקוח בעל אותו ת"ז.
```
void freeCustomerList();
```
הפונקציה משחררת את הזיכון שהוקצע עבור רשימת הלקוחות.

תאריך:

```
void initDate(Date* date);
```
הפונקציה מאתחלת תאריך.
```
int readDateFromFile(FILE* f, Date* d);
```
הפונקציה קוראת תאריך מהקובץ.
```
void printDateDataToFile(FILE* f, const Date* date, int fileType);
```
הפונקציה מדפיסה את נתוני התאריך לקובץ.
```
void printDate(const void* date);
```
הפונקציה מדפיסה את התאריך למסך.
```
int checkYear(int year);
```
הפונקציה בודקת תקינות השנה שנקלטה מהמשתמש .
```
int checkMonth(int month);
```
הפונקציה בודקת תקינות החודש שנקלט מהמשתמש .
```
int checkDay(int day, int month, int year);
```
הפונקציה בודקת תקינות היום שנקלט מהמשתמש .

```
int daysInMonth(int month, int year);
```
הפונקציה בודקת כמה ימים יש בחודש ספציפי בשנה ספציפית כדי לבדוק תקינות תאריך.

עובד:

```
int initNewEmployee(Employee* e);
```
הפונקציה מאתחלת עובד חדש.

```
int readEmployeeFromFile(FILE* f, Employee* e, int fileType);
```
הפונקציה קוראת את נתוני העובד מהקובץ.

```
employee_role getRoleFromUser();
```
הפונקציה קולטת תפקיד של עובד מהמשתמש.

```
void updateEmployeeSalary(const double salary, Employee* e);
```
הפונקציה מקבלת עובד וערך חדש של משכורת ומעדכנת את הנתון.

```
void printEmployee(const void* e);
```
הפונקציה מדפיסה את העובד למסך.

```
void printEmployeeDataToFile(FILE* f, const Employee* employee, int branchCode,
int fileType);
```
הפונקציה מדפיסה את נתוני העובד לקבצים.

```
int isEmployeeName(const Employee* e, const char name[]);
```
הפונקציה מחזירה 1 אם השם מתאים לעובד שהתקבל, 0 אם לא.

```
int isEmployeeID(const Employee* e, const int id);
```
הפונקציה מחזירה 1 אם הת"ז מתאים לעובד שהתקבל, 0 אם לא.

```
int isSameEmployee(const Employee* e1, const Employee* e2);
```
הפונקציה מקבלת שני עובדים ומשווה ביניהם, מחזירה 1 אם הם שווים.

```
void freeEmployee(const void* e);
```
הפונקציה משחררת את הזיכרון שהוקצע עבור מערך העובדים.

ניהול עובדים:

```
void initEmployeeManager(int fileType);
```
הפונקציה מאתחלת את מערך העובדים.

```
void continueEmployeeInit(FILE* f, Employee* e, int fileType);
```
הפונקציה מקבלת קובץ ומוסיפה את העובדים לפי הקובץ הנתון.

```
void printEmployeeManager();
```
הפונקציה מדפיסה את כל העובדים של הבנק למסך.

```
int addEmployee(const Employee* e, int state, int branchCode);
```
הפונקציה מוסיפה עובד חדש למערך העובדים.

```
Employee* findEmployeeByNameOrID(const char name[], const int id);
```
הפונקציה מחפשת עובד לפי שם או ת"ז ומחזירה אותו.

```
void sortEmployeeArray(sort_type sortType);
```
הפונקציה ממיינת את מערך העובדים.

```
sort_type getSortTypeFromUser();
```
הפונקציה מקבלת סוג מיון מהמשתמש.

```
int compareEmployeeByName(const void* employee1, const void* employee2);
```
הפונקציה מקבלת שני מצביעים לvoid, הופכת אותם לעובדים ומשווה אותם לפי השם, מחזירה 1 אם שווים.

```
int compareEmployeeByRole(const void* employee1, const void* employee2);
```
הפונקציה מקבלת שני מצביעים לvoid, הופכת אותם לעובדים ומשווה אותם לפי התפקיד,מחזירה 1 אם שווים.

```
int compareEmployeeBySalary(const void* employee1, const void* employee2);
```
הפונקציה מקבלת שני מצביעים לvoid, הופכת אותם לעובדים ומשווה אותם לפי המשכורת.

```
Employee* searchEmployee();
```
הפונקציה מבצעת חיפוש בינארי על מערך העובדים לפי השיטת מיון שנבחרה.

```
sort_type getSortType();
```
הפונקציה מחזירה את שיטת המיון הנוכחית.

```
void freeBankEmployeesArray();
```
הפונקציה משחררת את הזיכון שהוקצע עבור מערך העובדים.

```
int removeEmployee(const Employee* employee);
```
הפונקציה מוחקת עובד מהמערך ומהקובץ.

מידע אנושי:

```
void initHumanData(HumanData* hData, char humanType);
```

הפונקציה מאתחלת מידע על בן אדם.

```
int readHumanDataFromFile(FILE* f, HumanData* h, int fileType);
```

הפונקציה קוראת מידע מקובץ.

```
void printHumanDataToFile(FILE* f, const HumanData* human, int fileType);
```

הפונקציה מדפיסה מידע לקובץ.

```
void printHumanData(const void* hData);
```

הפונקציה מדפיסה מידע למסך.

פונקציות כלליות:

```
char* initString(char* buf, int size);
```

הפונקציה מאתחלת מחרוזת.

```
int compareTwoStringsNoCaseSensetive(char s1[], char s2[]);
```

הפונקציה משווה שני מחרוזות בלי להתייחס לאותיות גדולות או קטנות, מחזירה 1 אם שווים.

```
char* ltrim(char* s);
```

הפונקציה מוחקת רווחים משמאל למילה.

```
char* rtrim(char* s);
```

הפונקציה מוחקת רווחים מימין למילה.

```
char* trim(char* s);
```

הפונקציה מחזירה מחרוזת ללא רווחים מיותרים.

```
char* initDynamicString();
```

הפונקציה מאתחלת מחזורת דינאמית.

```
int initInteger(int min, int max);
```

הפונקציה מאתחלת מספר עם מספר ספרות בין min לmax.

```
void flush();
```

הפונקציה עושה ניקוי לבאפר.

```
void fileFlush(FILE* f);
```

הפונקציה עושה ניקוי לבאפר שנקלט מקובץ.

```
void generalFunction(void* arr, size_t size, size_t elem_size, void
(*func)(void*));
```

הפונקציה גנרית שמקבלת מערך מסוג void וקורת לפונקציה מסוג void ומבצעת פעולות על המערך.

רשימות:

```
int initList1(LIST1* list);
```

הפונקציה מאתחלת רשימה מסוג רשימה מקושרת חד כיוונית.

```
NODE1* insertToList1(NODE1* node, const void* value, size_t data_size);
```

הפונקציה מכניסה איבר בסוף הרשימה החד כיוונית.

```
int deleteFromList1(NODE1* pNode);
```

הפונקציה מוחקת איבר מרשימה חד כיוונית.

```
int freeList1(LIST1* pList);
```

הפונקציה משחררת את הזיכון שהוקצע עבור הרשימה החד כיוונית.

```
NODE1* findInList1(NODE1* pNode, void* Value);
```

הפונקציה מבצעת חיפוש איבר ברשימה חד כיוונית.

```
int initList2(NODE2* list);
```

הפונקציה מאתחלת רשימה מסוג רשימה מקושרת דו כיוונית.

```
NODE2* createNode2(Customer data);
```

הפונקציה בונה איבר חדש לרשימה ומאתחלת בו מידע מסוג לקוח.

```
void insertToEndList2(NODE2* node, Customer data);
```

הפונקציה מכניסה איבר לסוף הרשימה הדו כיוונית.

```
int deleteFromList2(NODE2* pNode);
```

הפונקציה מוחקת איבר מתוך הרשימה הדו כיוונית.

חלוקת אחריות

	קריסטינה	דוד
בניית קבצי c ו-header	✓	
בנייה ראשונית של כל מבני המערכת לפי התרשים	✓	
הקמת מערכים דינאמיים, הקצאות בזיכרון	✓	
מימוש הכנסת ומחיקת איברים ממערכים	✓	
שחרור הקצאות זיכרון ביציאה מהתוכנית	✓	
בניית רשימה מקושרת חד כיוונית ומימוש הפונקציות שלה	✓	
בניית רשימה מקושרת דו כיוונית ומימוש פונקציות שלה	✓	
בניית קבצי טקסט	✓	
בניית קבצים בינאריים		✓
קליטת נתונים מקבצי טקסט	✓	✓
קליטת נתונים מקבצים בינאריים	✓	✓
שמירת נתונים בקבצי טקסט		✓
שמירת נתונים בקבצים בינאריים		✓
מחיקת נתונים מקבצי טקסט	✓	✓
מחיקת נתונים מקבצים בינאריים		✓
מימוש פונקציה גנרית המקבלת מערך מסוג void*	✓	
בניית Macros		✓
מימוש פונקציות ראשיות בכל קבצי המערכת	✓	✓
מימוש פונקציות קליטת נתונים למערכים ורשימות מקושרות	✓	
מימוש פונקציות הדפסת נתונים למסך ממערכים ורשימות	✓	
מימוש פונקציות מחיקה ממערכים ורשימות מקושרות	✓	
החלטה על רכיבי התפריט הראשי ובנייתו	✓	✓
שמירה על קוד יעיל ומסודר	✓	✓
בדיקות מערכת	✓	✓
טיפול בהערות קומפילציה וקריסות מערכת	✓	✓
בניית דוח לפרויקט	✓	
הסבר על שמירה בקבצים בינאריים		✓
בניית שרטוט מערכת	✓	