

A Programozás Alapjai 1. Nagyházi Feladat

Felhasználói Dokumentáció

A program célja

Ez a program képes egy cég alkalmazottjainak adatait egyszerre kezelni és azokat egy Hash-Táblába betölteni, mellyel villámgyorsan lehet egy specifikus alkalmazottat megtalálni, annak az adatait módosítani, majd azokat újra lementeni fájlba.

A programot **konzolos felületen** keresztül lehet kezelni.

Adatok kezelése

A program a futás ideje alatt a betöltött adatokat a memóriában tárolja.

Ha a memóriában (5. menüpont) módosítjuk az adatakat, azok nem kerülnek mentésre a fájlban. A mentés az csak a 6. menüpont meghívásakor történik meg. Ha a programból kiléünk anélkül, hogy mentenénk, akkor az adatak elvesznek.

Bemenet

- A bemeneti formátum kötött, szükséges hogy CSV fájl legyen.
- Excelből való kimentés során a következő módon: CSV UTF-8 vesszővel tagolt
- Elválasztó jel a vessző: ,
- A fájlban nem lehetnek üres mezők, ha egy mező üres akkor azt szövegesen kell jelezni, vagy legalább egy - jelet oda tenni.
- A mezők sorrendje:

```
id,nev,szul_datum,nem,lakhely,email,telefon,szemelyi_szam,beosztas,reszleg  
,felettes,munkakezdet,munkavege,munkarend,bankszamla,fizetes
```

- Erre példa:

```
001,Kovács János,1985-10-  
23,Férfi,Budapest,kovacs.j@ceg.hu,06301234567,123456AB,Mérnök,IT,Nagy  
Főnök,08:00,16:00,H-P,11773333-12345678,500000
```

A program használata

Induláskor egy menü jelenik meg előttünk melynek opciói számozva vannak és a számuk beírásával majd Enter -t nyomva lehet őket meghívni.

Relatív elérési út: A programhoz képest számított elérési út, pl.: `adatok.csv` vagy `Dokumentumok\adatok.csv`

Abszolút elérési út: Teljes elérési út, pl.: `C:\Users\Nevem\Dokumentumok\adatok.csv`

Menüpontok részletezve:

0. Segítség kiírása

Újra megjeleníti a menülistát.

1. *Beolvasás csv fájlból

- **Működés:** Betölt egy a CSV-ben található adatokkal benépesített Hash-Táblát
- **Figyelem:** Ez a funkció **törli** a memoriában lévő előző táblázatot! Ha összefűzni szeretne két fájlt, használja először ezt, majd az 5-ös menüpont "Hozzáadás" funkcióját.
- **Teendő:** Írja be a fájl nevét (pl. `input.csv`) és nyomjon `Enter`-t.
- **Megjegyzés:** A program alapértelmezetten minden a saját könyvtárában keresi a megadott fájlt, ha máshol található akkor add meg vagy relatív vagy abszolút elérési út formájában.

2. Keresés a beolvasott táblában

- Egy konkrét dolgozó adatlapjának lekérése. A pontos azonosításhoz (ütközések elkerülése végett) a rendszer három adatot kér be egymás után:
 - **ID**
 - **E-Mail**
 - **Születési dátum**

3. Kilistázzuk a beolvasott táblát

- A képernyőre írja az összes betöltött alkalmazott nevét. Az egyes vödrökben a láncolást -> nyilakkal jelzi. Pl.: Molnár Ágoston -> Rácz Boglárka

4. Megnézzük hány ütközés történt beolvasás közben

- Technikai statisztika. Megmutatja, mennyire telített a tábla, és hány százalékban fordult elő "ütközés" (amikor két név hash kódja megegyezik). Felhasználói szempontból ez az adatbázis hatékonyságát jelzi.

5. Módosíthatjuk a táblát, beszúrhatunk, törölhetünk, adatokat változtathatunk meg

- Ez a szerkesztő almenü. A választáshoz írja be a számot:

1. Hozzáadás

- Manuálisan: A program bekéri egy alkalmazott adatait amelyet egyesével megadhatunk.
- Fájlból: Egy másik CSV fájl elérési útját kéri be a program, amelyből betölti az adatokat és eltárolja a jelenlegiek mellett.

2. Törlés: A törlendő személy azonosításához a program bekéri az ID-t, E-Mailt és Születési dátumot.

3. Módosítás:

- A program bekéri a három adatot (ID, E-Mail, Születési dátum) amellyel beazonosítható az alkalmazott
- A program kilistázza a mezőket számozva, majd a szám beírásával lehet kiválasztani melyiket szeretnénk felülírni
- Végül bekéri azt az adatot, amellyel felül szeretnénk írni a jelenlegit

6. Kiírhatjuk a módosított táblát egy új fájlba

- Az aktuális, memóriában lévő adatbázis kimentése.
- **Teendő:** Adjon meg egy fájlnevet (pl. export.csv)
- **Felülírás:** Ha a fájl már létezik, a program biztonsági kérdést tesz fel (1 = igen, 2 = nem).
- **Megjegyzés:** A program alapértelmezetten mindenig a saját könyvtárában keresi a megadott fájlt, ha másolható akkor add meg vagy relatív vagy abszolút elérési út formájában.

7. Bezárjuk a programot

- Bezárja az alkalmazást. A program rákérdez a megerősítésre. **Ne felejts el menteni (6-os gomb) kilépés előtt!**

A Programozás Alapjai 1. Nagyházi Feladat Programozási Dokumentációja

Bevezetés

Alkalmazott Nyilvántartó Rendszer (Hash-Táblával)

1. A projekt célja:

A projekt célja egy C nyelven írt, konzolos alkalmazás megvalósítása, melyel képes a felhasználó egy vállalat alkalmazottainak adatait külső fájlból betölteni, módosítani, lementeni. A rendszer egy Hash-Tábla épül amely biztosítja a kulcs ismeretében az egyes adatok gyors elérését, átlagosan **O(1)** idő alatt.

A szoftver lehetővé teszi a magyar karakterek helyes kezelését széles karakterek használatával (wchar_t), ezzel Excel export/import kompatibilissé téve a programot.

2. Modulok

A kód modulárisan van felépítve, a funkcionálitás több forrásfájlra (.c) és fejlécfájlra (.h) van bontva a karbantarthatóság érdekében.

Fájlstruktúra:

- main.c : A program belépési pontja. Kizárolag a főciklust (while(run)) és a magas szintű menüvezérlést tartalmazza.
- datastructs.h : A közösen használt adatstruktúrákat deklarálja

- `ht.c/ht.h` : A Hash-Tábla kezeléséhez szükséges függvényeket tartalmazza, memóriát foglal, beszűr, töröl.
- `io.c/io.h` : Input/Output műveletek. Fájlból/konzolról beolvas, fájlba kiír.
- `linkedlist.c/linkedlist.h` : Láncolt listához használt segédfüggvények. Fájlok beolvasásakor és a vödrökbe való beszúráskor is keletkezik egy láncolt lista.
- `fnv1a.c/fnv1a.h` : A Fowler-Noll-Vo (FNV1-a) hashelő algoritmus implementációja

Adatszerkezetek

Az adatok tárolása egy hierarchikus struktúrában történik a `datastruct.h`-ban definiáltak szerint.

Alkalmazott

A fő struktúra az `Alkalmazott` nevű struktúra, mely egyszerre tartalmaz minden adatot és működik láncolt lista elemként. Ezen felül képes tárolni egy már előre generált Hash-t a későbbi optimalizáció kedvéért.

Felépítése:

Minden `Alkalmazott` egy csomópont, amely tartalmaz 3 féle különböző adatot, egy saját magára hivatkozó mutatót, és egy 32 bites szám tárolására képes változót.

Személyes adatokra mutató pointer: `SzemelyesAdat *`

Munkavégzéssel kapcsolatos adatokra mutató pointer: `MunkaAdat *`

Pénzügyi adatokra mutató pointer: `PenzugyiAdat *`

Egy önhivatkozó pointer láncoláshoz: `struct Alkalmazott *kov`

Egy 32 bites szám a már létező hash eltárolására: `uint32_t storedHash`

Megjegyzés: minden karakter `wchar_t` karakter, implementációtól függően 2 vagy 4 bájt is lehet.

Személyes adat struktúra

Nyolc darab különböző személyes adatot tárol az alkalmazottról, és a kulcs is ezek közül van generálva.

ID: max 16 karakter // kulcs része

Név: max 64 karakter

Születési dátum: max 24 karakter, ÉÉÉÉ-HH-NN formátumban // kulcs része

Nem: max 16 karakter

Lakhely: max 64 karakter

E-Mail: max 64 karakter // kulcs része

Telefon: max 20 karakter

Személyi szám: max 16 karakter

Munka adat struktúra

Hat darab különböző munkavégzéssel kapcsolatos adatot tárol az alkalmazotról.

Beosztás: max 64 karakter

Részleg: max 64 karakter

Felettes: max 64 karakter

Munka kezdete: max 24 karakter, ÉÉÉÉ-HH-NN

Munka vége: max 24 karakter, ÉÉÉÉ-HH-NN

Munkarend: max 32 karakter

Pénzügyi adat struktúra

Csak két darab adatot tárol, egy bankszámlaszámot ha van és a fizetés mennyiségét

Bankszámlaszám: max 128 karakter

Fizetés: max 32 karakter

Hash-Tábla

Ez az adatszerkezet egy dinamikusan foglalt tömb mely Alkalmazott* elemeket tartalmaz, ezek a "vödrök" (buckets), és ezen kívül még annak a méretét.

Algoritmusok

Hashelés

A program az FNV-1a nem kriptográfiai algoritmus 32 bites változatát használja, mivel rövid sztringeken alkalmazva nagyon gyors és kevesebb eséllyel okoz ütközést. Az egyediség biztosítása érdekében a kulcs 3 féle adatból áll össze: Email + " " + Születési Dátum + " " + ID

Ütközések kezelése:

A vödrös hashelésnek köszönhetően ütközések esetén sincs probléma, hiszen minden vödörben egy láncolt lista található melynek ütközés esetén a végére van csatolva az új elem.

Ütközés esetén a program elejtől a végéig bejárja az indexen található láncolt listát és összehasonlíta a megadott elemet az azokban található elemekkel.

Módosítás

Ha a felhasználó egy olyan adatot szeretne módosítani amely szükséges a kulcs generálásához akkor az egész elem törlésre és újra beszűrésre kerül, mivel ezzel lehet elkerülni a hash változása esetén az elveszést.

Lépései:

1. Megnézi, hogy a módosított elem "kulcsfontosságú"-e
2. Ha igen, akkor létrehoz egy másolatot az eredetiről, abban módosítja az elemet, újra hashel, törli az eredetit, beszűrja a másolatot
3. Ha nem, akkor csak felülírja az elemet

Memóriakezelés

A program dinamikus memóriakezelést és a szivárgások észlelése érdekében debugmallocot használ.

Amikor nem fontos, hogy az elem mindenkinél kinullázva legyen, akkor malloc-ot használ, viszont amikor sztringeknek foglal helyet akkor a biztonság érdekében malloc-ot használ, hiszen így hiba esetén is üres sztringnek minősül.

Felszabadításra három különböző függvény van használatban

`freeNode` : Egy láncolt elem vagy al-struktúra felszabadítása

`linkedListFree` : Egy teljes láncolt lista felszabadítása

`htfree` : A teljes Hash-Tábla és minden tárolt elem felszabadítása a programból való kilépéskor vagy az előző felülírásakor.

Karakterkódolás

A C nyelven belül a magyar ékezetes betűk támogatása érdekében wchar_t karaktereket, azaz széles karakterekből álló sztringeket használ. Ezeket csak kiíráskor módosítja az UTF-8 kompatibilitás érdekében a wcstombs függvény segítségével több bájtból álló karakterekké.

Tervezői döntések

Elérési út megadása

- Ha a fájl a programfájl mellett van, akkor elég a neve, pl.: `adatok.csv`, de a legbiztosabb módszer az abszolút elérési út: `C:\Users\Nev\Documents\adatok.csv`

Hash-Tábla kulcsa

- A Hash-Tábla működési elve alapján szükséges egy kulcs ami kiszámíthatóvá teszi a táblán belüli elhelyezést. Ezt a kulcsot 3 adatból generáljuk, ID, E-Mail, és Születési Dátum.

Lefordítás és futtatás

A program C99 környezetben lett írva, CLion program segítségével és CMake alkalmazásával.

Lefordítás gcc segítségével:

```
gcc main.c ht.c utils.c linkedlist.c fnv1a.c -o NagyHF -Wall -Wextra
```

Szükséges könyvtárak

Saját:

- datastructs.h
- fnv1a.h
- ht.h
- io.h
- linkedlist.h
- utils.h

Beépített:

- stdio.h
- wchar.h
- locale.h
- stdbool.h
- stddef.h
- stdint.h
- inttypes.h

Külső:

- debugmalloc.h

Főbb függvények:

uint32_t FNV1a(const wchar_t *str)

Generál egy 32 bites számot a bekért sztringből

A bekért wchar_t karakterekből álló sztringet először átkonvertálja UTF-8 kompatibilis multibájt karakterekből álló sztringgé, majd azt bájtonként feldolgozza. Két konstanssal dolgozik, egy FNV offset és egy FNV prím értékkel. Első lépésként a kimenetet beállítja az offset értékére, majd minden a bekért szöveg minden bájtjával azt először XOR-olja majd beszorozza a prímmel. A végső érték egy 32 bites szám lesz amelyet visszaad a program.

Paraméterek

str - Ez egy pointer azokra a wchar_t karakterekből álló sztringre amelyből a hash készül.

Visszatérési érték

Bemenettől függetlenül mindig egy uint32_t típusú 32 bites számot ad vissza.

HashTable *htcreate(int initSize);

Létrehoz egy Hash-Táblát amellyel képes a htinsert() függvény dolgozni

Ez a program csak fix méretű Hash-Táblákkal tud dolgozni, ezért szükséges egy kezdeti méret megadása amely változatlan marad a futam végéig.

Paraméterek

initSize - A Hash-Tábla mérete.

Visszatérési érték

Egy pointert ad vissza a létrehozott Hash-Táblához.

int htinsert(HashTable *ht, Alkalmazott linkedListHead);**

Beszúrja egy láncolt lista elemeit a bekért Hash-Táblába

Végigmegy a láncolt lista elemein egyesével, mindegyikből csinál egy hasht a név, e-mail cím, és születési dátum felhasználásával, majd veszi a beadott Hash-Tábla méretét és azzal maradékosan leosztva a Hash-t megkapja, hogy melyik indexre kell beszúrni a kérte elemet. Abban az esetben, ha ütközés történik, akkor a vödrökön belül láncolja az elemeket.

Paraméterek

ht - Egy pointer a Hash-Táblára amelybe beszúrni kívánunk.

linkedListHead - Egy pointer egy láncolt lista első pointerjére, amelynek az elemeit beszúrnunk kívánunk.

Visszatérési érték

0 - minden rendeb

-1 - Láncolt lista mérete nagyobb mint a Hash-Tábla mérete.

-2 - Nem sikerült a kulcsot felépíteni a név, e-mail cím és születési dátum kombinációjából.

-3 - Nem sikerült egy új Alkalmazott elemet foglalni, amely szükséges a Hash-Táblába való beszúráshoz.

-4 - Nem sikerült az új Alkalmazott struktúra összes elemének memóriát foglalni.

-5 - Valamely a két paraméter közül NULL.

Megjegyzés:

- Az eredeti láncolt lista nem változik

void printHashTable(HashTable const *ht);

Kiíratja a hash tábla összes elemét

Végigmegy a hash tábla összes vödrén és kiírja az ott tárolt alkalmazottakat. Ütközés esetén a láncolt lista elemeit '->' jellet elválasztva jeleníti meg.

Paraméterek

ht - Pointer a kiíratni kívánt HashTable struktúrára

Visszatérési érték

void

Megjegyzés:

- Ha ht NULL akkor nem csinál semmit

void htfree(HashTable *ht);

Felszabadítja a Hash Tábla által foglalt memóriát.

Végigmegy az összes vödrön és felszabadítja először a bennük tárolt adatokat, magukat a vödröket, és végül a Hash-Táblát.

Paraméterek

ht - Pointer a felszabadítani kívánt ht struktúrára

Visszatérési érték

void

Megjegyzés

- Ha ht null, semmit sem csinál

int htresize(HashTable *ht);

Megkétszerezi a bekért tábla méretét

A bekért tábla vödreiről készít egy másolatot, egy olyan tömbbe amely mérete kétszerese az előzőnek, majd felülírja a Hash-Táblának a régi tömb mutatóját az újjal miután a régit felszabadította. A méretet is változtatja.

Paraméterek

ht - Egy mutató arra a Hash-Táblára amelynek a méretét szeretnénk megnövelni

Visszatérési érték

- Egy mutató az új, nagyobb táblára.

int htdelete(HashTable *ht*, Alkalmazott const target);

Töröl egy elemet a táblából

Megkeresi a célpontot a bekért táblában, majd kifűzi a láncból és felszabadítja.

Paraméterek

ht - A tábla melyből törölni szeretnénk

target - Az az elem melyet törölni szeretnénk

Visszatérési érték

0 - minden rendben

-1 - Valami probléma történt

bool htfind(HashTable const *ht*, Alkalmazott const target);

Megmondja, hogy egy elem megtalálható-e a táblában

A keresett elemet megkeresi a hash értékének kiszámításával majd ha egyezést talál a kulcsfontosságú elemeik között akkor igaz értéket ad vissza.

Paraméterek

ht - A tábla melyben keresni szeretnénk

target - A célpont melyet meg szeretnénk találni

Visszatérési érték

- Egy boole érték, mely megadja, hogy a keresés sikeres volt-e vagy sem

bool htupdate(HashTable *ht*, Alkalmazott *target*, int *fieldType*, wchar_t const **newValue*);

Egy elem mezőjének változtatására alkalmas függvény

A bekért táblában megkeresi a bekért célpontot, majd annak a megadott mezőjét felülírja a megadott új értékkel.

Paraméterek

ht - A tábla melynek szeretnénk az elemét változtatni

target - A célpont melyet változtatni szeretnénk

fieldType - A mező melyet változtatni szeretnénk

newValue - Az új érték mellyel felül szeretnénk írni a régit

Visszatérési érték

- Egy boole érték mely megadja a művelet sikerességét

Alkalmazott *inHt(HashTable const ht, Alkalmazott const *target);*

Megkeres egy elemet a táblában és visszaadja annak a mutatóját

Az bekért táblában megkeresi a bekért elemet és ha megtalálta akkor visszaad arra az elemre egy mutatót

Paraméterek

ht - A tábla melyet vizsgálunk

target - A célpont melyet keresünk

Visszatérési érték

- Egy Alkalmazott pointer a megtalált elemre

Megjegyzés

- Ha nem találta meg az elemet akkor a visszatérési érték NULL

Alkalmazott *readFromCSV(char const filePath);*

Egy CSV fájlból beolvassa az elemeket egy láncolt listába

A bekért elérési úton lévő CSV fájl minden sorából készít egy láncolt lista elemet amelyet belefűz egy dinamikusan foglalt láncolt listába majd a végén visszaadja annak az első elemére mutató pointert.

Paraméterek

filePath - A CSV fájl melyből be szeretnénk olvasni

Visszatérési érték

- Egy láncolt listára mutató pointer mely tartalmazza a CSV fájl sorait csomópontokként

char *readPath(void);

Beolvass a szabványos bemenetről egy elérési utat

Beolvass a szabványos bemenetről egy dinamikusan foglalt elérési utat.

Paraméterek

void

Visszatérési érték

- Egy elérési út

bool pathExists(const char *path);

Megvizsgálja, hogy a bekért elérési út létezik-e

Megpróbálja megnyitni a bekért elérési úton lévő fájlt, és ha sikeres volt akkor visszaad egy igaz értéket, ha nem, akkor egy hamisat.

Paraméterek

path - A vizsgált elérési út

Visszatérési érték

- Egy boole érték arról, hogy az elérési út létezik-e

Alkalmazott *readFromInput(void);

Bekér egy alkalmazottat a szabványos bemenetről

Egyetlen egy Alkalmazott elemet bekér a szabványos bemenetről a bekért mezők megadott értékeivel népesítve.

Paraméterek

void

Visszatérési érték

- Egy Alkalmazott pointer a bekért adatokkal feltöltve

int writeToCSV(HashTable ht, char const path);

Kiírja egy tábla tartalmát fájlba

A bekért tábla elemeit rendezetlenül kiírja a megadott elérési úton található fájlba, az elejére beszúrva egy BOM jelzést ezzel jelezve az UTF-8 kódolást az Excel-nek.

Paraméterek

ht - A tábla melynek elemeit ki szeretnénk írni

path - Az elérési út melybe írni szeretnénk

Visszatérési érték

0 - Ha minden rendben

-1 - Ha a paraméterek valamelyike nem elérhető

-2 - Ha nem sikerült megnyitni a fájlt

Alkalmazott *linkedListNodeCreate(SzemelyesAdat sz, MunkaAdat m, PenzugyiAdat p);*

Létrehoz egy új Alkalmazott elemet a bekért struktúrákból

A bekért struktúrákat belefűzi egy új láncolt lista elembe és a következő pointert NULL-ra állítja.

Paraméterek

sz - A személyes adatokat tartalmazó struktúrára egy mutató

m - A munkaügyi adatokat tartalmazó struktúrára egy mutató

p - A pénzügyi adatokat tartalmazó struktúrára egy mutató

Visszatérési érték

- Egy Alkalmazott struktúrára mutató pointer amely tartalmazza a bekért elemeket

void *linkedListAppend(Alkalmazott *head, Alkalmazott newElement);*

Beszűr egy új elemet a láncolt lista végére

Bekér egy láncolt lista mutatójára mutató pointert majd azt dereferálva annak a végére beszűrja a bekért elemet.

Paraméterek

head - A láncolt lista eleje, melybe beszúrni szeretnénk

newElement - Az elem melyet beszúrni szeretnénk

Visszatérési érték

void

Megjegyzés

- Ha sikertelen a beszúrás, vagy nem létezik valamelyik paraméter akkor NULL a visszatérési érték

void *freeNode(Alkalmazott *node);*

Felszabadít egy láncolt lista elemet

Egy egyedülálló láncolt lista elemet felszabadít.

Paraméterek

node Az elem melyet fel szeretnénk szabadítani

Visszatérési érték

void

void linkedListFree(Alkalmazott **head);

Felszabadít egy egész láncolt listát

Végigmegy az egész láncolt listán és minden elemet felszabadít egyesével.

Paraméterek

head - A láncolt lista melyet fel szeretnénk szabadítani

Visszatérési érték

void

int linkedListLen(Alkalmazott **head);

Megmondja, hogy egy láncolt lista milyen hosszú

A bekért láncolt lista mutatójára mutató pointert dereferálja, és azt bejárja, majd minden nem NULL érték találása esetén növel egy számlálót melyet a végén visszaad.

Paraméterek

head - A láncolt lista melynek hosszát vizsgáljuk

Visszatérési érték

- Egy int mely tartalmazza a lista méretét