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, mellyel képes a felhasználó egy vállalat alkalmazottainak adatait külső fájlból betölteni, módosítani, lementeni. A rendszer egy Hash-Táblára 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 funkcionalitás több forrásfájlra (.c) és fejlécfájlra (.h) van bontva a karbantarthatóság érdekében.

Fájlstruktúra:

- `main.c` : A program belépési pontja. Kizárólag 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, beszúr, 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 legenerált 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 alkalmazottról.

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 elejétől a végéig bejárja az indexen található láncolt listát és összehasonlítja 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 beszúrásra 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, beszúrja a másolatot
3. Ha nem, akkor csak felülírja az elemet

Memóriakezelés

A program dinamikus memóriakezelést és a szivárgások észlelése érdekében debugmalloc-ot használ.

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

Felszabadításra három különböző fügvény 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.

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
stdlib.h
string.h
wchar.h
locale.h
stdint.h
stdbool.h
inttypes.h

Külső:
debugmalloc.h
```