Hash-Tábla C nyelven

A program célja

Ez a program egy Hash-táblát **(tábla)** valósít meg a C nyelven belül. amely egy cég alkalmazottainak adatait képes tárolni.

A táblának képesnek kell lennie ezekre az alapvető funkciókra:

- Adatok beolvasása fájlból,
- Keresés, beszúrás, módosítás, törlés
- Adatok visszamentése fájlba

A tábla kulcsa egy sztring, amely az alkalmazott e-mail címéből, születési dátumából, és cégen belüli egyedi azonosítójából áll szóközökkel elválasztva, amely az **FNV-1a** algoritmussal egy 32 bites számmá lesz lehashelve.

Tárolt adatok szerkezete:

A program három féle struktúrában tárolja az alkalmazottakhoz releváns adatokat:

Személyes adatok:

```
- Cégen belüli egyedi azonosító    - Név    - Születési dátum    - Nem
- Lakhely    - E-mail cím    - Telefonszám    - Személyi szám#####
Munkaügyi adatok
- Munkaköri beosztás    - Részleg    - Felettes    - Munkavégzés kezdete
- Munkavégzés vége (ha van)    - Munkarend
```

Pénzügyi adatok

```
- Bankszámlaszám    - Fizetés
```

Ezekre a struktúrákra pointerek mutatnak amelyek egy **Alkalmazottak** nevű struktúrában tárolódnak el.

Példa:

```
typedef struct {
    char id[16];    char nev[64];    char szul_datum[11]; // (YYYY-MM-DD)
char nem[6];    char lakhely[64];    char email[64];    char telefon[20];
char szemelyi_szam[12];} SzemelyesAdat;

typedef struct {
    char beosztas[64];    char reszleg[64];    char felettes[64];    char
```

```
munkakezdet[11];    char munkavege[11];    char munkarend[32];} MunkaAdat;

typedef struct {
    char bankszamla[34];    double fizetes;} PenzugyiAdat;

typedef struct Alkalmazott {
    SzemelyesAdat *szemelyes_adatok;    MunkaAdat *munka_adatok;
PenzugyiAdat *penzugyi_adatok;    struct Alkalmazott *kov;  // láncolt
lista miatt} Alkalmazott;
```

Hashelő függvény: FNV-1a

A tábla a **Fowler-Noll-Vo 1a** nem kriptográfiai algoritmust használja a kulcsok kezelésére.

Az FNV-1a algoritmusnak is egy specifikusabb, azaz a 32 bites számokat előállító verzióját használom. Ez egy egyszerű kódrészlettel demonstrálva így néz ki:

```
uint32_t FNV1a(const wchar_t *str) {
    const size_t size = wcstombs(NULL, str, 0);
    if (size == (size_t)-1){
        return 1;
    }

    char *utf8 = malloc(size+1);
    if (!utf8) return 1;

    wcstombs(utf8, str, size + 1);

    uint32_t hash = FNV_OFFSET; // előre definiált konstans 2166136261u
      for (size_t i = 0; i < size; i++) {
        hash ^= (uint8_t)utf8[i];
        hash *= FNV_PRIME; // előre definiált konstans 16777619u
    }    free(utf8);
return hash;  ```

A következő pseudokód leírja a táblába való beszúrás menetét
```

1. kulcs = (email && szuletesi datum && id) -> UTF-8
2. kulcs -> FNV-1a -> uint32_t
3. index = uint32_t % ht_size
4. ht[index] = Alkalmazott ```

Ütközéskezelés

Mivel a hashtábla alap működési elve alapján az ütközések elég nagy adatmennyiség mellett teljesen elkerülhetetlenek, ezért ezeknek a kezelésére vödrös hash módszert alkalmazunk. Minden kulcson belül az értékek egy láncolt listában vannak eltárolva, az új

elemek a lista végére kerülnek beszúrásra. Amikor kereséskor egy indexre érkezünk akkor a program lineárisan bejárja ezeket.

Fájlkezelés

Mivel a programnak képesnek kell lennie:

- fájlból beolvasni
- memórián belül azt módosítani
- fájlba visszaírni
  ezért a fájl formátuma egy egyszerű CSV lesz a könnyedség kedvéért, hiszen Excel fájlokat is könnyedén lehet CSV-be átkonvertálni és dolgozni is egyszerű vele.

Beolvasáskor a program dinamikusan kér memóriát az operációs rendszertől, hogy el tudja tárolni abban az adatokat.

A táblán elvégezhető műveletek

A program csak 6 szimpla alapművelet elvégzésére kell, hogy képes legyen:

1. beszúrás: **htinsert(Alkalmazott *alkalmazott)** függvény -> bool sikeres-e
2. keresés: **htlookup(Alkalmazott *alkalmazott)** függvény -> bool létezik-e ; int hol (index)
3. törlés: **htdel()** függvény -> bool sikeres-e
4. módosítás: **htupdate()** függvény -> bool sikeres-e, mi változott mire
5. fájlműveletek elvégzése: **htsave() / htload()** függvény -> bool sikeres-e