Programozás házi feladat

Illyés Dávid Gyula – ZYTGYT

A program két fájlal dolgozik az indulástól kezdve, az egyik fájlban termékek vannak felsorolva, azonosítóval, árral és névvel, a másik fájlban eladások kosarainak tartalma van felsorolva úgy, hogy minden sor egy vásárlás tartalmát reprezentálja, és ilyen sorokból akármennyi lehet a fájlban. A program először végig megy a vásárlások fájlon és minden elemnél (egy egész szám, ami egy termék azonosítója) végig megy a termékek fájl tartalmán és minden elemnél összehasonlítja az épp vizsgált azonosítót a fájlban lévő összes termék azonosítójával, ha egyezést talál akkor egy függvény használatával hozzáadja egy termékek (termek_lista) típusú hátulról strázsált láncolt listához, ez addig megy amíg a sor végére ér. Amikor eléri a sor végét, a most elkészített termek_lista -t hozzáadja egy szintén hátulról strázsált vásárlások (vasarlasok) típusú láncolt listához. Ezt addig ismételi amíg a vásárlások fájl végére nem ér. A beolvasás után egy függvénnyel kiszámolja minden korsár teljes értékét, és bele írja a vásárlások (vasarlasok) típusú lista elem sum_price változójába. Miután és csak is miután lefutott a summázó függvény, kiszámolja az összes vásárlás teljes értékének az átlagát. Miután ez a két függvény lefutott megkeresi az átlaghoz legközelebbi summázott kosárértéket, és kiírja az adott kosár teljes értékét és a kosár tartalmát.

A program által feldolgozott fileokban lévő adatszerkezetek így néznek ki:

- Termékek fájl minden sora egy terméket jelent, a sorokban pedig az id (egész szám) szóköz, ár (lebegő pontos szám) szóköz, név (karakter [150]), szintaktika alapján vannak az adatok, (termekek.txt) példa:

```
1 500.0 husleves
2 15000.0 gérvágó
3 1000.0 kukazsák
4 533.5 levegő
5 748.4 hell energia ital
6 6584.23 laptop
7 6139.345 kávé
8 23.2 dianás cukor
9 2374.34 műzli
10 1245.245 pohár
11 8399.45 víz
12 12345.843 föld
```

- Vásárlások fájl sorai a különböző vásárlások tételeinek id -jeivel van tele, minden sor végén van egy lezáró „n” karakter, egy sorban maximum 1000 elem van a lezáró karaktert beleszámolva, a sorok száma nincs limitálva, (vasarlasok.txt) példa:

```
1 5 3 6 8 42 n
2 8 4 16 27 n
2 4 8 10 12 n
1 3 5 7 9 11 n
```

Összefoglalva

termekek struct

```
typedef struct termekek {
    char nev[150];
    double ar;
    struct termekek *next;
} termekek, *termekek_ptr;
```

vasarlasok struct

```
typedef struct vasarlasok {
    double sum_price;
    struct vasarlasok *next;
    struct termekek *termek_lista;
} vasarlasok, *vasarlasok_ptr;
```

A program az „stdio.h”, „stdlib.h”, „string.h”, „math.h” könyvtárakat használja. A tényleges fájlból listába beolvasás előtt definiálva vannak a termékeket, illetve vásárlásokat listába helyező és listákból elemeket törlő funkciók. Az alábbiakban ezek vannak leírva:

push_product (új terméket láncol egy hátulról strázsált listába):

A függvény egy már létező terméklista fejét, a termék nevét és a termék árát kéri paraméterként és az új lista kezdőelem pointerét fogja visszaadni. Első lépésként a funkció foglal egy termekek lista elem méretű helyet a memóriában, majd aztán ellenőrzi, hogy a visszakapott pointer nem NULL pointer-e, mely esetben egy hibaüzenetet ír ki („Couldn’t allocate memory for new list item”) és kilép. Amennyiben sikerült memóriát foglalni, az új lista elembe bemásolja a paraméterben kapott nevet és árat, a „nev” és „ar” változóba. Ezután az eddigi lista fejére mutató pointert maga után láncolja a next pointerrel és megváltoztatja a lista fejének pointerét a most foglalt új lista elem pointerére, majd ezt visszaadja.

```
termekek_ptr push_product(termekek_ptr head, char nev[150], double ar) {
    termekek_ptr tp = malloc(sizeof(termekek));
    if (tp == NULL) {
        printf("Couldn\'t allocate memory for new list item\n");
        return 0;
    }
    strcpy(tp->nev, nev);
    tp->ar = ar;
    tp->next = head;
    head = tp;
    return head;
}
```

push_purchase (új eladást láncol egy hátulról strázsált listába):

A függvény egy már létező hátulról strázsált vasarlasok és termekek típusú pointert kér úgy, hogy mind a kettő az adott lista fejére mutat. A függvény először lefoglal egy új vasarlasok lista elem méretű memória helyet és le ellenőrzi, hogy sikerült-e lefoglalni (a visszakapott pointer nem NULL pointer), amennyiben NULL pointer, hiba üzenetet ír ki („Couldn’t allocate memory for new list item”) és kilép, ha nem az fut tovább. Az új lista elem next pointerének oda adja a paraméterként kapott előző lista elem fejét. A funkció meghívásakor

megadott termekek típusú lista fejének pointerére megváltoztatja az új lista elem termek_lista pointerét. Majd az új lista elemet teszi a lista fejévé és ezt visszaadja.

```
vasarlasok_ptr push_purchase(vasarlasok_ptr head, termekek_ptr termek_p) {
    vasarlasok_ptr vp = malloc(sizeof(vasarlasok));
    if (vp == NULL) {
        printf("Couldn\'t allocate memory for new list item\n");
        return 0;
    }
    vp->next = head;
    vp->termek_lista = termek_p;
    head = vp;
    return head;
}
```

delete_products (minden az adott termék listában lévő elemet felszabadít):

A funkció nem rendelkezik visszatérési értékkel és paraméterként egy termekek típusú lista fejének pointerét kéri. Ellenőzi, hogy nem-e NULL értékű a „head”, ami azt jelezné, hogy semmi nincs benne, majd amennyiben ez nem igaz egy while loopban csinál egy ideiglenes („tmp” nevű) termekek típusú pointert, amibe eltárolja a következő lista elem fejének a címét, hogy az ne vesszen el amíg a jelenlegit felszabadítjuk. Ezt követően felszabadítjuk a jelenlegi lista elemet és az előzőleg elmentett következő elemet a lista fejévé tesszük, hogy a következő ciklusban az legyen felszabadítva.

```
void delete_products(termekek_ptr head) {
    if (head != NULL) {
        while (head == NULL) {
            termekek_ptr tmp = head->next;
            free(head);
            head = tmp;
        }
    }
}
```

delete_purchases (minden az adott vásárlások listában lévő elemet felszabadít beleértve az összes „termek_lista” -t):

Egy visszatérési érték nélküli függvény, ami egy vasarlasok típusú láncolt lista fejének pointerét kéri. Megnézi, hogy nem e NULL pointer a függvény feje, ha ez nem igaz egy while ciklusban addig megy amíg a head pointer nem lesz NULL értékű. Minden ciklusban egy ideiglenes vasarlasok típusú pointerben eltárolja a maga után következő lista elem pointerét, hogy az ne vesszen el addig amíg a jelenlegi felszabadul. Meghívja az előzőleg leírt delete_products függvényt, aminek paraméterként a jelenlegi lista elem termek_lista pointerét adja. Miután az lefutott free -eli a lista elem fejét és megváltoztatja az ideiglenes változóban eltárolt pointerre a lista fejének pointerét.

```
void delete_purchases(vasarlasok_ptr head) {
    if (head != NULL) {
        while (head == NULL) {
            vasarlasok_ptr tmp = head->next;
            delete_products(head->termek_lista);
            free(head);
            head = tmp;
        }
    }
}
```

sum_price (minden kosár teljes értékének kiszámolása külön- külön):

A függvény a vásárlások lista fejének pointerét kéri paraméterként (vasarlasok_prt), ez alapján végig megy fésű gerincének összes „fogán” (termek_lista), mindegyiknél összeadja az összes termék árát egy ideiglenes double változóba majd, amikor a lista végére ér, az summázott értékre megváltoztatja az adott gerinc elem sum_price változójának értékét. Ezt minden gerinc elemnél megcsinálja, amíg a lista végére nem ér. A függvénynek nincs visszatérési értéke.

```
void sum_price(vasarlasok_ptr head) {
    double sum = 0.0;
    for (vasarlasok_ptr v = head; v != NULL; v = v->next) {
        for (termekek_ptr t = v->termek_lista; t != NULL; t = t->next) {
            sum += t->ar;
        }
        v->sum_price = sum;
        sum = 0.0;
    }
}
```

avg_price (minden termék átlag összege):

A függvény egy vasarlasok típusú pointert kér paraméterként, ez alapján végig fut az összes gerinc elemen, eközben minden elemnél hozzáad egyet, egy darab számláló int típusú változóhoz, a gerincben lévő sum_price értékeket egy double típusú sum nevű változóba összeadja, amikor a lista végére ér, a summázott értéket elosztja a darabszámmal. A visszatérési értéke egy double, ami az átlagolás eredménye. Megjegyzés: az avg_price függvényt csak is a sum_price függvény meghívása után szabad meghívni, mivel anélkül nincsenek summázott kosárértékek, amivel ki tudná számolni az átlagot!

```
double avg_price(vasarlasok_ptr head) {
    double sum = 0, avg;
    int i = 0;
    for (vasarlasok_ptr v = head; v != NULL; v = v->next) {
        i++;
        sum += v->sum_price;
    }
    avg = (sum/i);
    return avg;
}
```

closest_to_avg (az átlaghoz legközelebbi kosárérték):

A függvény paramértere egy vasarlasok típusú pointer, ami a lista fejére mutat, végig megy a lista gerincén és minden elemnél megvizsgálja, hogy a teljes átlag (avg_price függvény által vissza adott érték) nagyobb-e, mint a jelenlegi a gerincben található lista elem sum_price értéke. Ez alapján a nagyobból kivonja a kissebbet, majd megnézi, hogy az így kapott érték kisebb-e, mint az előző ez egy ideiglenes változóban van tárolva (ha ez az első elem a gerincben és emiatt az ideiglenes változó az alap nulla értékén van akkor azt felülírja), ha kisebb akkor elmenti az ahhoz a vásárláshoz tartozó termek_lista pointert és a vásárlás értékét egy-egy ideiglenes pointerbe. Amikor a lista végére ér kiírja az átlaghoz legközelebbi kosár értékét és ki listázza az adott kosár tartalmát. Visszatérési értéke nincs a függvénynek.

```c
void closest_to_avg(vasarlasok_ptr head) {
    double curr_closest = 0;
    termekek_ptr out;
    vasarlasok_ptr value;
    double avg = avg_price(head);
    for (vasarlasok_ptr tmp = head; tmp != NULL; tmp = tmp->next) {
        if (avg < tmp->sum_price) {
            double r = fabs(tmp->sum_price - avg);
            if (curr_closest == 0) {
                curr_closest = r;
                out = tmp->termek_lista;
                value = tmp;
            } else if (curr_closest > r) {
                curr_closest = r;
                out = tmp->termek_lista;
                value = tmp;
            }
        } else if (avg > tmp->sum_price) {
            double r = fabs(avg - tmp->sum_price);
            if (curr_closest == 0) {
                curr_closest = r;
                out = tmp->termek_lista;
                value = tmp;
            } else if (curr_closest > r) {
                curr_closest = r;
                out = tmp->termek_lista;
                value = tmp;
            }
        }
    }
    printf("Az átlag kosárértékhez legközelebbi kosár értéke: %0.2f Ft,
tartalma:\n", value->sum_price);
    for (out; out->next != NULL; out = out->next) {
        printf("Termék név: %s, \f\tár: %0.2f Ft\n", out->nev, out->ar);
    }
}
```

file_to_list (fájlokból beolvassa az adatokat a listákba):
A függvény egy előre elkészített vasarlasok típusú lista fejét és a vásárlások, illetve a termékek fileok neveit kéri a visszatérési értéke pedig a már feltöltött lista gerincének fejére mutató pointer. Első lépésként megnyitja mind a kettő fájlt a paraméterként megadott nevek alapján és ellenőrzi, hogy sikeresen megnyílt e mind a kettő (nem NULL pointer a visszatérési érték). Majd deklarál néhány ideiglenes változót, amit később fog használni. Egy while ciklusban elkezd beolvasni sorokat a vásárlások fájlból, amíg a fájl végére nem ér. A cikluson belül azonnal létrehoz egy hátulról strázsált termekek típusú lista elemet, amibe aztán majd bele írja a beolvasott elemeket. Minden beolvasott sort elemeire bontja a szóközöknél majd minden elemnél (termék azonosítónál) egy while ciklussal végig fut a termékek fájl tartalmán sorról sorra. Minden terméknél összehasonlítja az azonosítót az épp keresett azonosítóval, amit a vásárlások fájl vizsgált sorából kibontott. Amennyiben egyezés van a sorban lévő adatokat (név, ár) oda adja a push_product függvénynek az imént létrehozott lista fejének pointerével együtt és kilép a ciklusból, majd pedig a termékek fájl épp olvasott sorát visszaállítja a legelső sorra. Ezt addig ismételi amíg a sor végére nem ér a vásárlások fájlban, ahol pedig oda adja a most elkészült termék lista fejét a push_purchase

függvénynek a paraméterként kapott vasarlasok típusú lista fejével együtt. Amikor a vásárlások fájl végére ér, bezárja mind a kettő fájlt és visszaadja a feltöltött lista fejének pointerét.

```
vasarlasok_ptr file_to_list(vasarlasok_ptr head, char purchase_file[], char
product_file[]) {
///******************/// FILEOK MEGNYITÁSA ///*****************///
    FILE *purchase_p = fopen(purchase_file, "r");
    if (purchase_p == NULL) {
        printf("Couldn\'t open purchases file\n");
        return 0;
    }

    FILE *product_p = fopen(product_file, "r");
    if (product_p == NULL) {
        printf("Couldn\'t open products file\n");
        return 0;
    }
////////////////////////////////////////////////////////////////////

    int tmp_id, eq;
    double tmp_ar;
    char tmp_nev[150];
    char tmp_line[1000], split[] = " ";
    while (fgets(tmp_line, sizeof(tmp_line), purchase_p) != EOF) {
        termekek_ptr product_list = malloc(sizeof(termekek));
        product_list->next = NULL;
        char *char_int = strtok(tmp_line, split);
        eq = strcmp(char_int, "n\n");
        while (char_int != NULL && eq != 0) {
            while (fscanf(product_p, "%d %lf %[^\n]%*c", &tmp_id, &tmp_ar,
tmp_nev) != EOF) {
                if (tmp_id == atoi(char_int)) {
                    product_list = push_product(product_list, tmp_nev, tmp_ar);
                    break;
                }
            }
            char_int = strtok(NULL, split);
            if (char_int == NULL)
                break;
            else
                eq = strcmp(char_int, "n\n");
            rewind(product_p);
        }
    if (char_int == NULL)
        break;
    head = push_purchase(head, product_list);
    }
    fclose(purchase_p);
    fclose(product_p);
    return head;
}
```

A fő program:
Itt létrehozunk egy vasarlasok típusú pointert és foglalunk egy vasarlasok lista elem méretű helyet a memóriában, ellenőrzi, hogy sikerült-e lefoglalni, ha sikerült beállítja a hátsó strázsát és halad tovább, ha nem sikerült hibaüzenetet dob ("Couldn’t allocate memory for purchases”) és kilép. Meghívja a file_to_list függvényt, aminek paraméterként a most elkészített lista fejét és a két fájl

nevét adja. Meghívja a sum_price függvényt, majd kiírja az átlag árat az avg_price függvény segítségével. A closest_to_avg függvény szintén meghívásra kerül a lista fejével paraméterként. Majd pedig a kellemetlenségek elkerülése végett meghívja a delete_purchases függvényt, ami felszabadítja a lista elemeit, ezután pedig kilép a program.

```
int main(void) {
    vasarlasok_ptr vp = calloc(1, sizeof(vasarlasok));
    if (vp == NULL) {
        printf("Couldn\'t allocate memory for purchases\n");
        return 0;
    }
    vp->next = NULL;
    vp = file_to_list(vp, "vasarlasok.txt", "termekek.txt");

    sum_price(vp);
    printf("A kosarak átlag ára %0.2f Ft\n\n", avg_price(vp));
    closest_to_avg(vp);

    delete_purchases(vp);
    return 0;
}
```

Tesztelés:
Először a listákat feltöltő (push_product és push_purchase) függvényeket teszteltem úgy, hogy soronként hívtam meg először a push_product függvény(eke)t, majd a push_purchase függvényt és manuálisan adtam meg nekik a paramétereket. A push_product függvény kezdetben mindig egyel eleinte nem láncolta hozzá az előző lista elemet, amit egy szimpla elírás okozott, így gyorsan ki lett javítva. Ez után a kettő közül egyik függvénnyel se adódott gond. Az alábbi sorokban a tesztelésre használt próba sorok vannak.

```
head_product = push_product(head_product, "tej", 1500.00);
head_product = push_product(head_product, "karaj", 430.23);
head_product = push_product(head_product, "utca lámpa", 123.45);
head_product = push_product(head_product, "kenyér", 23.23);
head_purchases =  push_purchase(head_purchases, head_product);
```

Miután ezek a függvények megbízhatóan működtek megírtam az ezeket a listákat felszabadító függvényeket (delete_products, delete_purchases), ezek az előző tesztben leírt próba listákat megfelelően felszabadították.

Ezután a sum_price függvényt próbáltam ki az előbbiekhez hasonló próba adatokkal. Itt eleinte az volt a gond, hogy túlment a lista végén. A problémát az okozta, hogy arra tesztelt a for ciklus, ami végig ment a lista elemeken, hogy a lista jelenlegi eleme (head) nem-e egyenlő NULL pointer, hanem arra, hogy a következő utáni (head->next) nem-e NULL pointer. Ez szintén gyorsan kijavításra került azzal, hogy head -re javítottam a hibás feltételt.

Az avg_price függvényt eleinte úgy teszteltem, hogy oda adtam neki a teljes adag adatot, amit majd a kész programnak tudnia kellett feldolgozni. Ezzel nem akadt semmi problémája Ezután a vásárlások fájlból kitöröltem az első sor kivételével minden mást. Így 0 -át adott vissza, ami nem volt megfelelő, de ez nem az avg_price függvény hibája volt. A gond az volt hogy a fő függvényben (main() -ben) calloc() helyett malloc() -ot használtam és így nem megfelelően lett lefoglalva az induló lista elem ami hibát okozott. Ez a gyors átírás után elmulasztotta a problémát.

A file_to_list függvény írása/tesztelése közben adódott a legtöbb probléma (ami meg lett oldva) a függvény első tesztjénél a ciklus nem ért véget és mindig egy végtelenített ciklus lett belőle. A problémát az okozta, hogy a sorokat lezáró n karakterrel való egyezést (while (beolvasott != 'n'))magában nem akarta érzékelni, mint ciklus futási feltétel, így mindig a végtelenségig ment. Erre az lett a megoldás, hogy csináltam egy külön változót, ami az értékét egy strcmp() függvénytől kapja, a függvény kimenete 0-tól eltérő szám, ha a kettő megadott string nem egyezik és 0 ha egyeznek, ezért így most már a while ciklusban arra is kellett tesztelnem, hogy addig menjen a ciklus amíg ez a változó nem egyenlő 0-val és amig az éppen keresett termék azonosítót tartalmazó változó nem lesz egyenlő NULL pointerrel. Ez egy kis időre megoldotta a problémát, de kreált egy újat, ami abból eredt, hogy ha az strcmp() -nek NULL pointert adunk meg paraméterként akkor a függvény hibát ad vissza. Ez egy if -el lett megoldva, ami megnézi, hogy a változó (ami lehet egy termék azonosító vagy n karakter vagy a sor végén egy NULL pointer) nem-e egyenlő egy NULL pointerrel, amennyiben az akkor kilépünk a ciklusból, ha nem akkor tovább mehet a program és az strcmp() megkaphatja. És ezzel az is járt, hogy így most már a 'fő' tehát a vásárlásokon végig menő ciklus után is kellett egy if -es ellenőrzést rakni, ami azt vizsgálja, hogy a változó, ami a termék azonosítót vagy n karaktert tartalmazza nem-e egyenlő NULL pointerrel, ha igen lépjünk ki mert a fájl végére értünk.