Éttermi Rendszer - Programozói Dokumentáció

1. Fejlesztői Környezet és Projekt Felépítése

1.1. Fejlesztői Környezet Követelmények

- Operációs rendszer: Linux vagy Windows
- Fordító: GCC 9.0 vagy újabb
- Szükséges fordítási paraméterek: `-Wall -Werror`
- Szükséges külső könyvtárak: `debugmalloc.h` (mellékelve)

1.2. Fordítási Útmutató

```
# Fordítás parancssorból
gcc -Wall -Werror asztalfoglalas.c asztalkezelo.c asztalok.c file.c main.c menu.c
rendelesek.c szamla.c terkep.c utils.c -o etterem

# Debugmalloc használatával
gcc -DMEMTRACE -Wall -Werror asztalfoglalas.c asztalkezelo.c asztalok.c file.c main.c menu.c
rendelesek.c szamla.c terkep.c utils.c -o etterem
```

1.3. Projekt Struktúra

A projekt moduláris felépítésű, az alábbi fájlokkal:

> #### Főbb forrásfájlok és szerepük
>
> - **main.c/h**
> - Program belépési pont
> - Főmenü implementáció
> - Program állapot kezelése
> - **asztalok.c/h**
> - Asztalok kezelése láncolt listában
> - Asztal foglalási logika
> - Asztal pozícionálás és elrendezés

- **menu.c/h**
    - Étlap kezelése dinamikus adatszerkezetben
    - Menüelemek hozzáadása/törlése
    - Ár és név kezelés
- **rendelesek.c/h**
    - Rendelések nyilvántartása
    - Rendelési tételek kezelése
    - Rendelés állapot követése
- **file.c/h**
    - Adatok perzisztens tárolása
    - Fájl műveletek és hibakezelés
    - Bináris fájlformátum kezelése
- **utils.c/h**
    - Általános segédfüggvények
    - Input validáció
    - Hibaüzenetek kezelése

2. Adatszerkezetek és Tervezési Megfontolások

2.1. Központi Adatszerkezetek

Program Állapot

```
typedef struct {
    AsztalLista* asztalok;    // Dinamikus asztal nyilvántartás
    MenuLista* menu;          // Dinamikus étlap kezelés
    RendelesLista* rendelesek; // Aktív rendelések
} ProgramAllapot;
```

Ez a struktúra a program központi állapottárolója. Tervezési megfontolások:

- Pointer típusú mezők a dinamikus memóriakezelés érdekében
- Moduláris felépítés - minden funkcionális egység külön kezelhető
- Egységbe zárt állapotkezelés - könnyű paraméterátadás

Asztal Kezelés

```
typedef struct Asztal {
    int azonosito;           // Egyedi azonosító
    int ferohely;           // Ülőhelyek száma
    int x, y;               // Pozíció a térképen
    int foglalt;            // Foglaltsági állapot
    struct Asztal* kov;     // Láncolt lista következő elem
} Asztal;
```

Tervezési megfontolások az asztalok kezeléséhez:

- Láncolt lista implementáció a dinamikus bővíthetőség érdekében
- Koordináta rendszer a vizuális megjelenítéshez
- Minimális memóriaigény - csak a szükséges adatok tárolása

Menü Kezelés

```
typedef struct MenuElem {
    char* nev;           // Dinamikus karakterlánc
    int ar;                 // Egységár
    struct MenuElem* kov;   // Következő menüelem
} MenuElem;
```

A menü kezelés tervezési szempontjai:

- Dinamikus karakterlánc a változó hosszúságú nevek miatt
- Láncolt lista a könnyű bővíthetőség érdekében
- Egyszerű árazási modell

2.2. Fájl Formátumok

asztalok.dat

```
Rekord méret: 17 byte
[4 byte] - azonosito (int)
```

```
[4 byte] - ferohely (int)
[4 byte] - x koordináta (int)
[4 byte] - y koordináta (int)
[1 byte] - foglalt állapot (char)
```

menu.dat

```
Változó hosszúságú rekordok
[1 byte] - név hossza (unsigned char)
[n byte] - név karakterei
[4 byte] - ár (int)
```

rendelesek.dat

```
Rekord mérete változó, a rendelési tételek számától függően.
[4 byte] - asztal azonosító (int)
[4 byte] - tétel száma (int)
Tételek ismétlődő rekordjai:
    [4 byte] - étel azonosító (int)
    [4 byte] - mennyiség (int)
```

3. Függvény Dokumentáció

3.1. Asztal Kezelő Függvények

asztalListaLetrehozas

```
AsztalLista* asztalListaLetrehozas(void);
```

Feladat: Új, üres asztal lista létrehozása

Visszatérés: Az új lista pointere, vagy NULL hiba esetén

Hibakezelés: Memória foglalási hiba esetén NULL visszatérés

asztalListaFelszabadit

```
void asztalListaFelszabadit(AsztalLista* lista);
```

Feladat: Asztal lista és minden elemének felszabadítása

Paraméterek: `lista` - A felszabadítandó lista (nem lehet NULL)

Mellékhatások: A lista és minden eleme felszabadításra kerül

3.2. Rendelés Kezelő Függvények

rendelesTetelHozzaad

```
int rendelesTetelHozzaad(Rendeles* rendeles, int menuElemAzonosito, int mennyiseg);
```

Feladat: Új tétel hozzáadása egy rendeléshez

Paraméterek:

- `rendeles` - A rendelés, amihez hozzáadjuk (nem lehet NULL)
- `menuElemAzonosito` - A rendelt étel azonosítója
- `mennyiseg` - A rendelt mennyiség (pozitív egész)

Visszatérés: 1 siker esetén, 0 hiba esetén

Hibakezelés:

- NULL pointer ellenőrzés
- Érvénytelen azonosító ellenőrzése
- Memória foglalási hiba kezelése

4. Hibakezelési Stratégiák

4.1. Általános Elvek

- Minden memóriafoglalás ellenőrzése
- Fájlműveletek hibakezelése
- NULL pointer ellenőrzések
- Érvénytelen paraméterek szűrése

4.2. Fájlkezelési Hibák

- Fájl megnyitási hibák kezelése
- Írási/olvasási műveletek ellenőrzése
- Fájl integritás ellenőrzése
- Automatikus fájl lezárás hiba esetén

4.3. Memóriakezelés

A program a `debugmalloc.h` könyvtárat használja a memóriaszivárgások és hibák detektálására:

- Minden allokált memória felszabadításra kerül
- Double-free hibák elkerülése
- NULL pointerek kezelése
- Memóriaszivárgások detektálása