Slovenská Technická Univerzita v Bratislave

Fakulta informatiky a informačných technológií

Správca pamäti

Zadanie 1

Predmet: Dátové štruktúry a algoritmy

Obdobie: Letný semester 2019/2020

Cvičiaci: Ing. Dominik Macko , PhD.

Študent: Bc. František Gič

Obsah

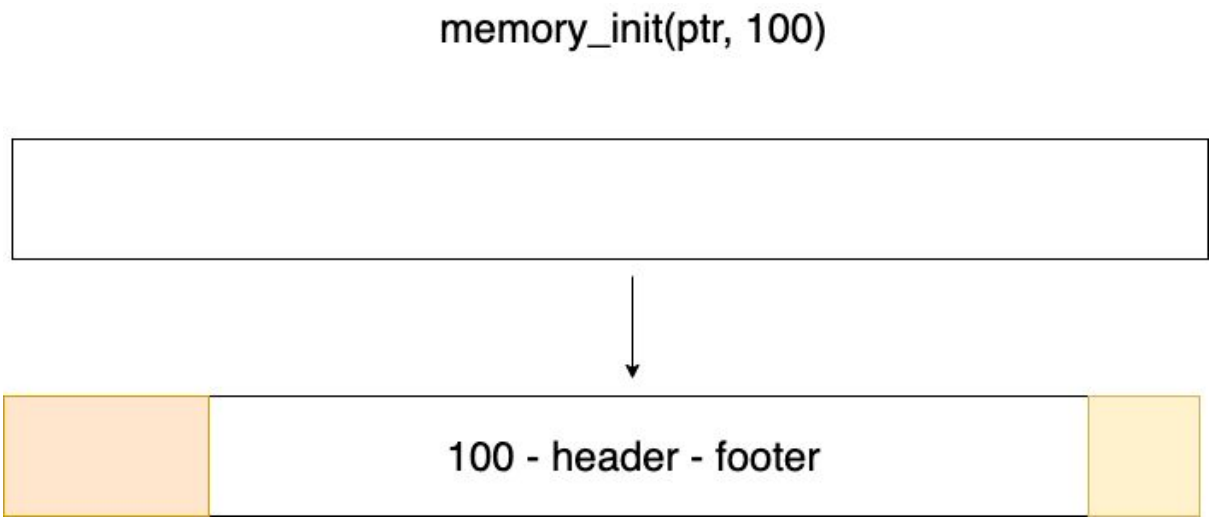
Úvod

Riešenie ktoré som vypracoval je inšpirované metódou explicitných zoznamov voľných pamätí s určitými zmenami podľa vlastného uváženia.
Použil som metódu globálnej hlavičky pamäte - “globálnej”, pretože na túto štruktúru ukazuje jediná globálna premenná programu počas celého runtime, hlavičky a pätičky (ďalej len *header* a *footer*) jednotlivých blokov pamäte.
Metóda má niekoľko možných vylepšení - napríklad zoradenie zoznamov voľnej pamäte podľa veľkostí, alebo použitie rozdielnej hlavičky pre voľné a alokované bloky - nezaimplementovaných z časových dôvodov a zadaní na iné predmety.

Implementácia

Inicializácia pamäte

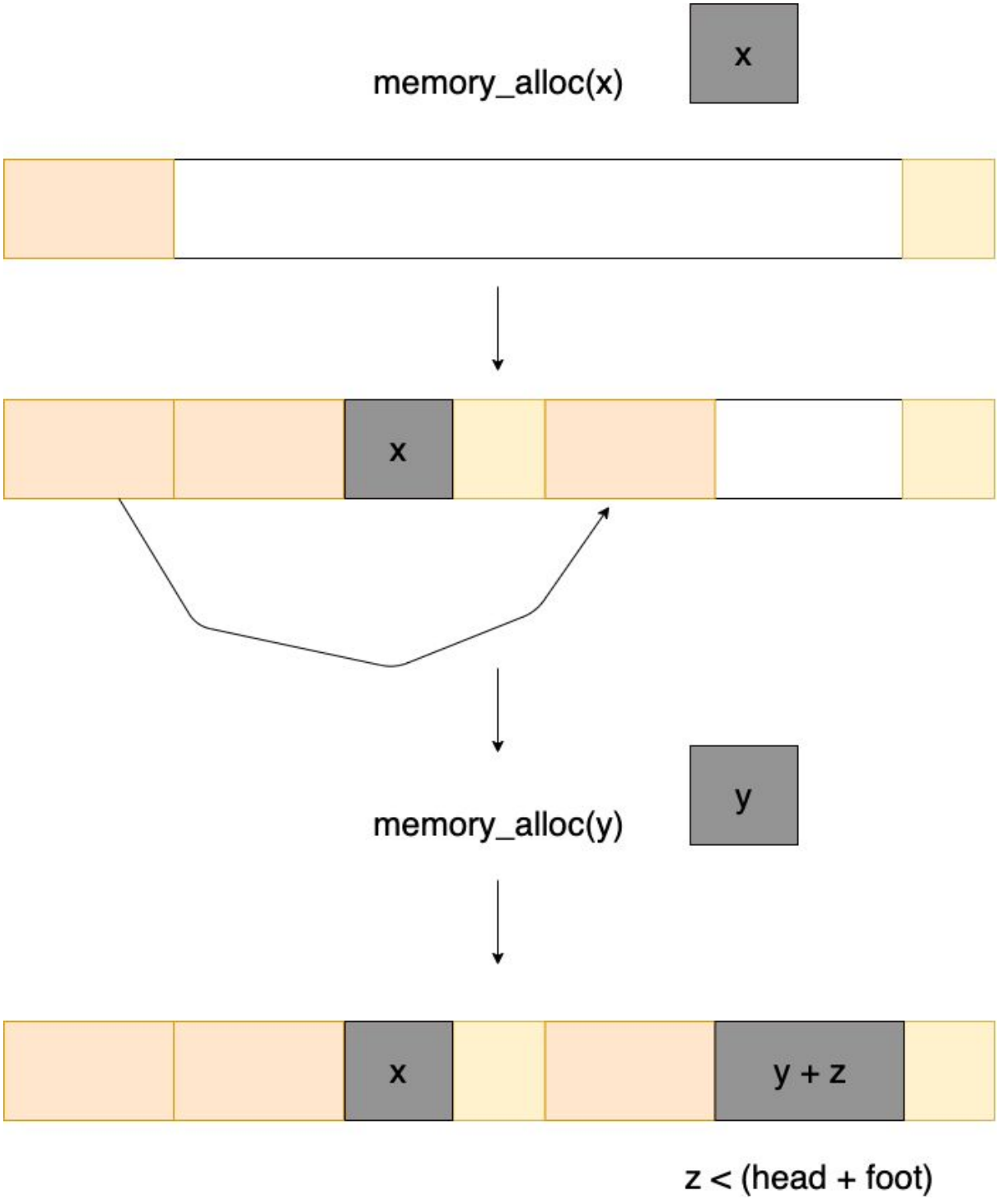
Inicializácia pamäte je pomerne jednoduchá. Funkcia skontroluje či pointer neukazuje na null, a či je veľkosť pamäte dostatočná na vytvorenie alespoň hlavičky a footeru.
V prípade, že sú tieto podmienky splnené, nastaví globálny pointer na danú pamäť, vyplní hlavičku, nastaví ju ako voľnú pamäť a nastaví do headeru aj do footeru vnútornú veľkosť pamäte - ktorá je menšia ako veľkosť v parametri, práve o veľkosť týchto dvoch štruktúr.



Obrázok 1: Diagram inicializácie pamäte

Alokácia

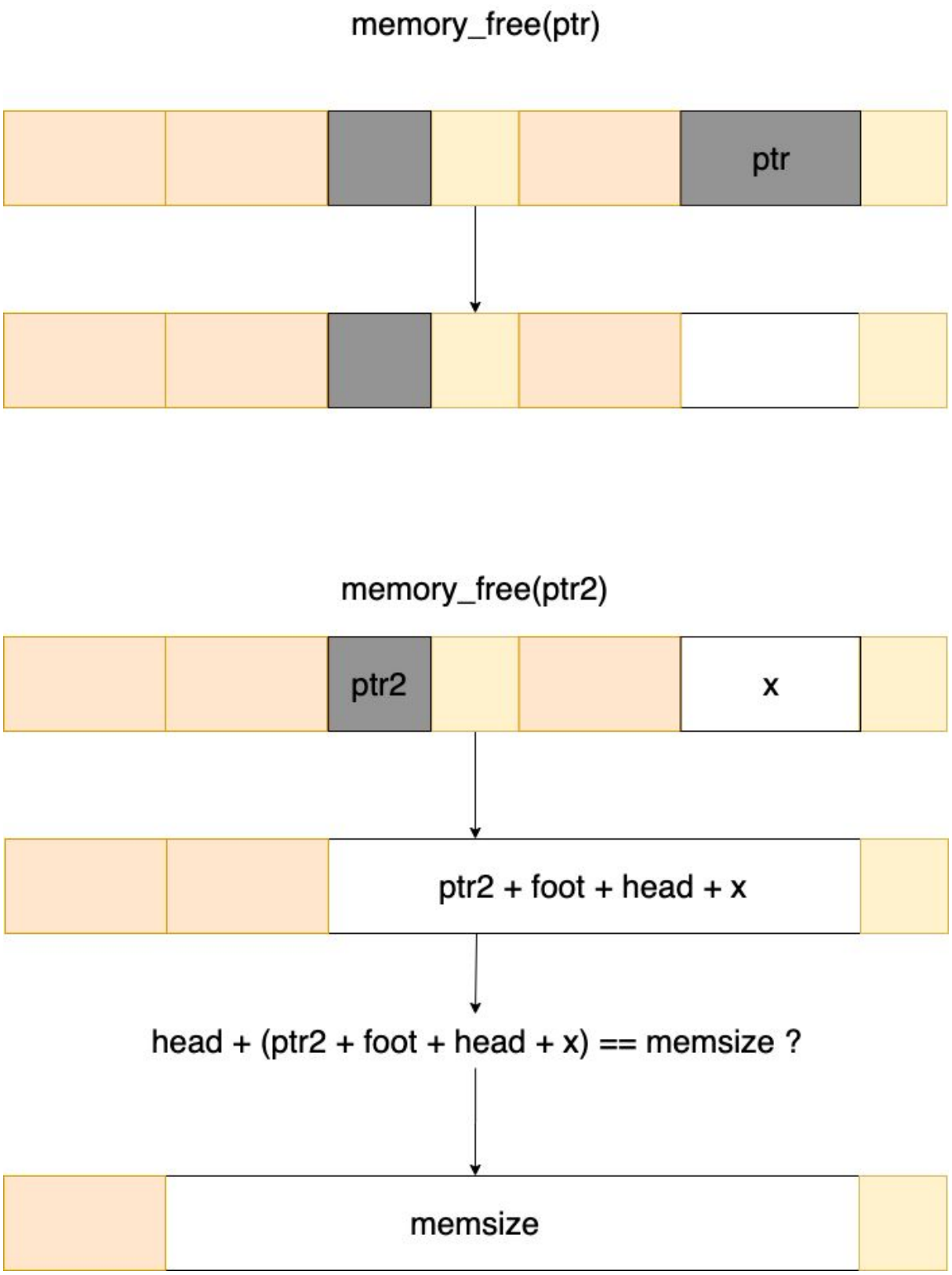
Alokácia pamäte funguje nasledovným spôsobom. Program traverzuje poľom prázdnych častí (chunkov) pamäte pomocou pointerov v každej hlavičke. Pokiaľ daný blok vyhovuje veľkosťou, je vhodným kandidátom na alokáciu.
Štandardná alokácia spočíva v tom, že preberá header daného bloku, označí sa ako alokovaný a vytvorí sa nový footer a header, posunutý o želanú veľkosť.
Tu sa nachádza jedno z mojich vylepšení - štandardne, pokiaľ sa po alokovaní pamäť voľného bloku nedokáže rozdeliť - pamäť nie je dostačujúca na nový header a footer a minimálne jeden byte - vráti sa celý chunk - to znamená, v krajných prípadoch býva vrátený blok ktorého veľkosť môže prevyšovať želanú veľkosť.
Jedným z ďalších krajných prípadov je, pokiaľ je daná voľná pamäť hneď na začiatku pamäte. V tomto prípade nemôžeme prebrať header - pretože je to globálny header. V tomto prípade nasleduje vytvorenie headeru posunutého o veľkosť globálneho headeru.
Do headeru alokovaného bloku sa vpíšu detaily o alokácií a veľkosti, a novovytvorený header sa nadpojí do zoznamu voľnej pamäte.
Nakoniec, užívateľovi sa vráti pointer ukazujúci na začiatok pamäťovej časti chunku.



Obrázok 2: Diagram alokácie pamäte

Uvoľňovanie pamäte

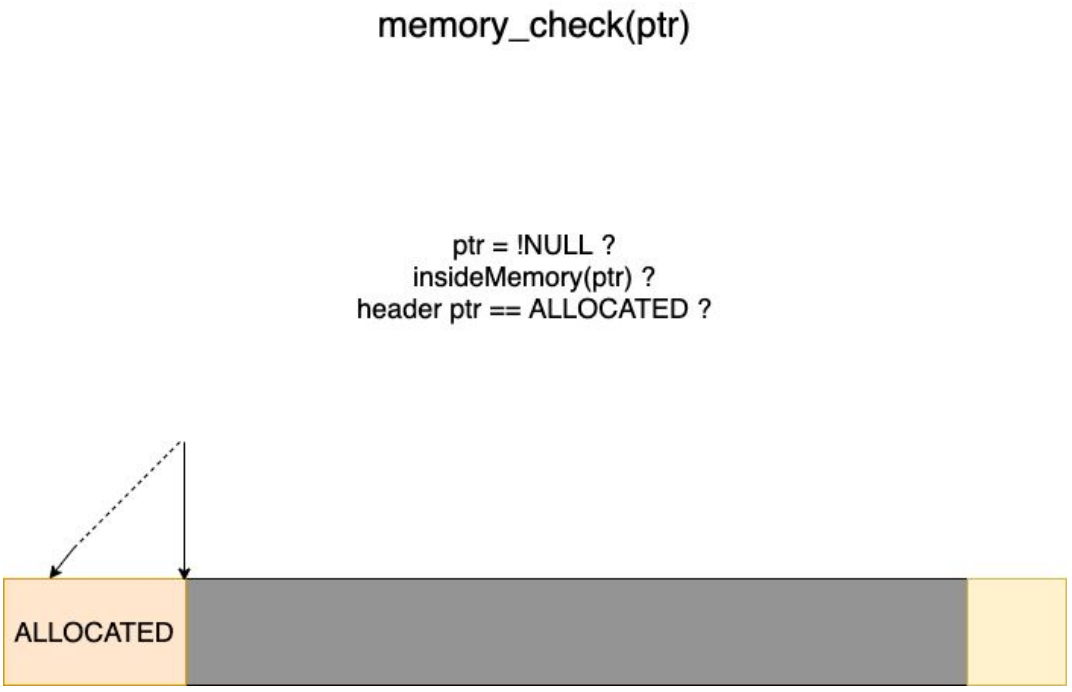
Uvoľňovanie je najkomplexnejšia funkcia tejto štvorice. Funkcia v sebe obsahuje metódu spájania blokov (merging chunks).
Po skontrolovaní validity headeru bloku pamäte funkciou *memory_check* funkcia kontroluje možný chunk bezprostredne za a pred danou pamäťou. Pokiaľ niečo takéto existuje, funkcia spojí dané bloky.
V prípade voľného bloku za pamäťou vyradí z činnosti header za pamäťou, označí aktuálny ako voľný a spočíta veľkosti týchto daných pamätí (vrátane footeru aktuálnej a headeru nasledovnej). Taktiež, nadpojí danú pamäť na miesto, kde bola nadpojená nasledujúca.
V prípade voľného bloku pred pamäťou, spočíta veľkosť predchádzajúceho, a tohto bloku a zapíše danú veľkosť do predchádzajúceho bloku.
Následne v prípade, že predchádzajúci blok ukazuje na konkrétny blok (čo sa môže stať iba v prípade, že predchádzajúci blok ukazoval predtým na nasledujúci blok po našom konkrétnom bloku - čiže prebehla aj akcia spájania vpred), tak zmeníme pointer, aby náš konkrétny blok preskočil.
Vylepšenie z mojej strany je znižovanie vnútornej fragmentácie, a to, konkrétne po uvoľnení daného bloku traverzovanie zoznamom voľných pamätí. Pokiaľ je pamäť príliš fragmentovaná, ale je kompletne prázdna (čiže súčet všetkých headerov a footerov a pamäťových blokov je vlastne veľkosť celej pamäte), tak sa celý zoznam voľných pamätí zahodí, a nechá sa len globálny pointer = pamäť v rovnakom stave, ako po inicializácií



Obrázok 3: Diagram uvoľňovania pamäte

Kontrola smerníka

Kontrola smerníka je jednoduchou funkciou. Predpokladáme že smerník smeruje na tú časť pamäte, ktorú vrátila funkcia memory_alloc.
Pre istotu však skontrolujeme, či náhodou pointer nie je NULL, alebo mimo regiónu ktorý bol inicializovaný. Pokiaľ je všetko v poriadku, pozrieme sa na header daného smerníka, a vyčítame či je alokovaný. V inom prípade vraciame 0.



Obrázok 4: Diagram kontroly pamäte

Testovanie

Test 1

Prideľovanie rovnakých blokov malej veľkosti (veľkosti 8 až 24 bytov) pri použití malých celkových blokov pre správcu pamäte (do 50 bytov, do 100 bytov, do 200 bytov)

```
TEST 1.1
Range [5 - 5] (bytes)
Memory size: 50 bytes
Memory size (within global header and footer): 30 bytes
Allocated blocks (count): 1
Allocated (requested): 5/30 bytes (16.67%)
Allocated (actual): 30/30 bytes (100.00%) (60.00% of total size)
Memory used by misc (footers and headers): 20/50 bytes (40.00%)
Empty memory: 0/30 bytes (0.00%)
```

```
TEST 1.2
Range [5 - 5] (bytes)
Memory size: 100 bytes
Memory size (within global header and footer): 80 bytes
Allocated blocks (count): 3
Allocated (requested): 15/80 bytes (18.75%)
Allocated (actual): 24/80 bytes (30.00%) (24.00% of total size)
Memory used by misc (footers and headers): 76/100 bytes (76.00%)
Empty memory: 0/80 bytes (0.00%)
```

```
TEST 1.3
Range [8 - 8] (bytes)
Memory size: 100 bytes
Memory size (within global header and footer): 80 bytes
Allocated blocks (count): 3
Allocated (requested): 24/80 bytes (30.00%)
Allocated (actual): 24/80 bytes (30.00%) (24.00% of total size)
Memory used by misc (footers and headers): 76/100 bytes (76.00%)
Empty memory: 0/80 bytes (0.00%)
```

```
TEST 1.4
Range [8 - 8] (bytes)
Memory size: 200 bytes
Memory size (within global header and footer): 180 bytes
Allocated blocks (count): 6
Allocated (requested): 48/180 bytes (26.67%)
Allocated (actual): 64/180 bytes (35.56%) (32.00% of total size)
Memory used by misc (footers and headers): 136/200 bytes (68.00%)
Empty memory: 0/180 bytes (0.00%)
```

```
TEST 1.5
Range [15 - 15] (bytes)
Memory size: 200 bytes
Memory size (within global header and footer): 180 bytes
Allocated blocks (count): 5
Allocated (requested): 75/180 bytes (41.67%)
Allocated (actual): 84/180 bytes (46.67%) (42.00% of total size)
Memory used by misc (footers and headers): 116/200 bytes (58.00%)
Empty memory: 0/180 bytes (0.00%)
```

```
TEST 1.6
Range [24 - 24] (bytes)
Memory size: 200 bytes
Memory size (within global header and footer): 180 bytes
Allocated blocks (count): 4
Allocated (requested): 96/180 bytes (53.33%)
Allocated (actual): 104/180 bytes (57.78%) (52.00% of total size)
Memory used by misc (footers and headers): 96/200 bytes (48.00%)
Empty memory: 0/180 bytes (0.00%)
```

Test 2

Prideľovanie nerovnakých blokov malej veľkosti (náhodné veľkosti 8 až 24 bytov) pri použití malých celkových blokov pre správcu pamäte (do 50 bytov, do 100 bytov, do 200 bytov)

```
TEST 2.1
Range [8 - 24] (bytes)
Memory size: 50 bytes
Memory size (within global header and footer): 30 bytes
Allocated blocks (count): 1
Allocated (requested): 19/30 bytes (63.33%)
Allocated (actual): 30/30 bytes (100.00%) (60.00% of total size)
Memory used by misc (footers and headers): 20/50 bytes (40.00%)
Empty memory: 0/30 bytes (0.00%)
```

```
TEST 2.1
Range [8 - 24] (bytes)
Memory size: 100 bytes
Memory size (within global header and footer): 80 bytes
Allocated blocks (count): 2
Allocated (requested): 29/80 bytes (36.25%)
Allocated (actual): 44/80 bytes (55.00%) (44.00% of total size)
Memory used by misc (footers and headers): 56/100 bytes (56.00%)
Empty memory: 0/80 bytes (0.00%)
```

```
TEST 2.1
Range [8 - 24] (bytes)
Memory size: 200 bytes
Memory size (within global header and footer): 180 bytes
Allocated blocks (count): 5
Allocated (requested): 67/180 bytes (37.22%)
Allocated (actual): 84/180 bytes (46.67%) (42.00% of total size)
Memory used by misc (footers and headers): 116/200 bytes (58.00%)
Empty memory: 0/180 bytes (0.00%)
```

Test 3

Prideľovanie nerovnakých blokov väčšej veľkosti (veľkosti 500 až 5000 bytov) pri použití väčších celkových blokov pre správcu pamäte (aspoň veľkosti 1000 bytov),

```
TEST 3.1
Range [500 - 5000] (bytes)
Memory size: 1000 bytes
Memory size (within global header and footer): 980 bytes
Allocated blocks (count): 1
Allocated (requested): 813/980 bytes (82.96%)
Allocated (actual): 813/980 bytes (82.96%) (81.30% of total size)
Memory used by misc (footers and headers): 56/1000 bytes (5.60%)
Empty memory: 131/980 bytes (13.37%)
```

```
TEST 2.2
Range [500 - 5000] (bytes)
Memory size: 1000 bytes
Memory size (within global header and footer): 980 bytes
Allocated blocks (count): 1
Allocated (requested): 883/980 bytes (90.10%)
Allocated (actual): 883/980 bytes (90.10%) (88.30% of total size)
Memory used by misc (footers and headers): 56/1000 bytes (5.60%)
Empty memory: 61/980 bytes (6.22%)
```

```
TEST 3.3
Range [500 - 5000] (bytes)
Memory size: 4000 bytes
Memory size (within global header and footer): 3980 bytes
Allocated blocks (count): 2
Allocated (requested): 3387/3980 bytes (85.10%)
Allocated (actual): 3387/3980 bytes (85.10%) (84.68% of total size)
Memory used by misc (footers and headers): 76/4000 bytes (1.90%)
Empty memory: 537/3980 bytes (13.49%)
```

Poznámka: Za povšimnutie stojí tentokrát voľná pamäť. Doteraz bývala nulová, z dôvodu že alokovaná (requested) pamäť nikdy neprekračovala celkovú veľkosť pamäte.. Kdežto pokiaľ sa vygeneruje číslo väčšie ako celková pamäť, program skončí. (loop kým memory_alloc nevrátil NULL). V prípade predchádzajúcich testov sa vyplnil posledný blok na maximum a až potom program skončil.

Test 4

Prideľovanie nerovnakých blokov malých a veľkých veľkostí (veľkosti od 8 bytov do 50 000) pri použití väčších celkových blokov pre správcu pamäte (aspoň veľkosti 1000 bytov).

```
TEST 4.1
Range [8 - 50000] (bytes)
Memory size: 100000 bytes
Memory size (within global header and footer): 99980 bytes
Allocated blocks (count): 3
Allocated (requested): 83081/99980 bytes (83.10%)
Allocated (actual): 83081/99980 bytes (83.10%) (83.08% of total size)
Memory used by misc (footers and headers): 96/100000 bytes (0.10%)
Empty memory: 16823/99980 bytes (16.83%)
```

```
TEST 4.2
Range [8 - 50000] (bytes)
Memory size: 100000 bytes
Memory size (within global header and footer): 99980 bytes
Allocated blocks (count): 5
Allocated (requested): 93678/99980 bytes (93.70%)
Allocated (actual): 93678/99980 bytes (93.70%) (93.68% of total size)
Memory used by misc (footers and headers): 136/100000 bytes (0.14%)
Empty memory: 6186/99980 bytes (6.19%)
```

```
TEST 4.3
Range [8 - 50000] (bytes)
Memory size: 50000 bytes
Memory size (within global header and footer): 49980 bytes
Allocated blocks (count): 3
Allocated (requested): 49889/49980 bytes (99.82%)
Allocated (actual): 49889/49980 bytes (99.82%) (99.78% of total size)
Memory used by misc (footers and headers): 96/50000 bytes (0.19%)
Empty memory: 15/49980 bytes (0.03%)
```

Záver

Toto cvičenie bolo asi najzaujímavejšie ktoré som zatial na FIIT vypracovával (možno porovnateľné s PPGSO v C++ v 3.roč). Toto cvičenie ma stálo veľa potu ale dalo mi veľmi veľa. Hlavne ohľadom správy pamäte, pointerovej aritmetiky a pod, keďže v praxi sa viac zaoberám vyššími programovacími jazykmi.

Z testovania som si odniesol nasledovné:

1. Naučte sa pracovať s debuggerom 😃
2. *(footer *)((char*) actual) + size+ sizeof(header)*
   !=
   *(footer *) ((char*) actual + size +sizeof(header))*

Z testovania vyplýva nasledovné:

Dôležitým faktorom pri malých pamätiach je optimalizácia pamäte ktorá je nepotrebná pre samotnú pamäť ktorú alokujeme, ale pre správu pamäte. To je vidieť pri malých pamätiach (do 50B), kde headery a footery (štruktúry potrebné pre správu pamäte) zaberajú cca 50% celej pamäte. Riešením pre tento problém, tak ako som už sa spomína v úvode je možno používanie ukazovateľa, či je daná pamäť alokovaná bitový posun, ako bol v prednáške. Taktiež, použitie hlavičky s pointerom na ďalšiu je zbytočné pri hlavičke alokovaného bloku - v prípade explicitných zoznamov voľnej pamäte.