




9/3/2021

# Správca pamäti

DSA – Zadanie 1



Samuel Hetteš  
ID: 110968  
STU FIIT 2020/2021

## OBSAH

Zadanie – správca pamäti .....	3
Implementácia .....	4
• Štruktúra pamäťových blokov .....	4
✓ Voľný blok .....	4
✓ Alokovaný blok .....	5
• Opis funkcií .....	5
✓ Memory init .....	5
✓ Memory alloc .....	6
✓ Memory free .....	6
✓ Memory check .....	8
Testovanie .....	9
• Testovanie zo zadania .....	9
• Vlastné testy .....	12
• Testovanie efektívnosti .....	15
Záver .....	16

## ZADANIE – SPRÁVCA PAMÄTI

V štandardnej knižnici jazyka C sú pre alokáciu a uvoľnenie pamäti k dispozícii funkcie `malloc` a `free`. V tomto zadaní je úlohou implementovať vlastnú verziu alokácie pamäti.

Konkrétnejšie je vašou úlohou implementovať v programovacom jazyku C nasledovné ŠTYRI funkcie:

- `void *memory_alloc(unsigned int size);`
- `int memory_free(void *valid_ptr);`
- `int memory_check(void *ptr);`
- `void memory_init(void *ptr, unsigned int size);`

Vo vlastnej implementácii môžete definovať aj iné pomocné funkcie ako vyššie spomenuté, nesmiete však použiť existujúce funkcie `malloc` a `free`.

Funkcia `memory_alloc` má poskytovať služby analogické štandardnému `malloc`. Teda, vstupné parametre sú veľkosť požadovaného súvislého bloku pamäte a funkcia mu vráti: ukazovateľ na úspešne alokovaný kus voľnej pamäte, ktorý sa vyhradil, alebo `NULL`, keď nie je možné súvislú pamäť požadovanej veľkosti vyhradiť.

Funkcia `memory_free` slúži na uvoľnenie vyhradeného bloku pamäti, podobne ako funkcia `free`. Funkcia vráti `0`, ak sa podarilo (funkcia zbehla úspešne) uvoľniť blok pamäti, inak vráti `1`. Môžete predpokladať, že parameter bude vždy platný ukazovateľ, vrátený z predchádzajúcich volaní funkcie `memory_alloc`, ktorý ešte nebol uvoľnený.

Funkcia `memory_check` slúži na skontrolovanie, či parameter (ukazovateľ) je platný ukazovateľ, ktorý bol v nejakom z predchádzajúcich volaní vrátený funkciou `memory_alloc` a zatiaľ nebol uvoľnený funkciou `memory_free`. Funkcia vráti `0`, ak je ukazovateľ neplatný, inak vráti `1`.

Funkcia `memory_init` slúži na inicializáciu spravovanej voľnej pamäte. Predpokladajte, že funkcia sa volá práve raz pred všetkými inými volaniami `memory_alloc`, `memory_free` a `memory_check`. Viď testovanie nižšie. Ako vstupný parameter funkcie príde blok pamäte, ktorú môžete použiť pre organizovanie a aj pridelenie voľnej pamäte. Vaše funkcie nemôžu používať

globálne premenné okrem jednej globálnej premennej na zapamätanie ukazovateľa na pamäť, ktorá vstupuje do funkcie `memory_init`. Ukazovatele, ktoré prideliť vaša funkcia `memory_alloc` musia byť výhradne z bloku pamäte, ktorá bola pridelená funkcii `memory_init`.

## IMPLEMENTÁCIA

- **metóda:** obojsmerne spájaný explicitný zoznam blokov

### Štruktúra pamäťových blokov

- **veľkosti blokov:** zaokrúhľovanie na násobok 4, kvôli lepšej manipulácii
- posledný bit určuje či je blok alokovaný alebo voľný
- 0 – voľný blok, 1 – alokovaný blok

#### ✓ VOĽNÝ BLOK

HEADER			PAYLOAD	FOOTER
unsigned int size	HEADER *next	HEADER *previous		unsigned int size
4B	8B	8B		4B

- unsigned int size = veľkosť payloadu + veľkosť ukazovateľov, keďže pri alokovanom bloku neuchováame žiadne ukazovatele
- HEADER \*next = pointer na ďalší voľný blok
- HEADER \*previous = pointer na predchádzajúci voľný blok

## ✓ ALOKOVANÝ BLOK

FULL_BLOCK_HEADER	PAYLOAD	FOOTER
unsigned int size		unsigned int size
4B		4B

- unsigned int size = veľkosť payload

## + Opis funkcií

### ✓ MEMORY INIT

- **časová zložitosť** =  $O(n)$  - memset
- **pamäťová zložitosť** =  $O(1)$
- **Postup:**
  - globálny ukazovateľ sa nastaví na region pointer, ktorý príde do funkcie ako parameter
  - zaokrúhlenie veľkosti pamäte na najbližší nižší násobok 4. Zvyšné byty ostnú nevyužité.
  - vytvorenie hlavnej hlavičky

MAIN_HEADER	
unsigned int size	HEADER *start
4B	8B

- unsigned int size = veľkosť pamäte - veľkosť hlavnej hlavičky
- HEADER \*start = ukazovateľ na začiatkový voľný blok
- vytvorenie hlavičky a pätičky prvého voľného bloku
- označenie voľných bytov

## ✓ MEMORY ALLOC

- **časová zložitosť:**  $O(n + m)$  - počet voľných blokov, memset
- **pamäťová zložitosť:**  $O(1)$
- **Postup:**
  - zaokrúhlenie veľkosti na najbližší násobok 4
  - nájdenie voľného bloku: **FIRST FIT**
  - ak blok nebol nájdený vráti NULL
  - spojenie predchádzajúcej a nasledujúcej hlavičky
- **Rozdelenie bloku:**
  - ak požadovaná veľkosť je rovnaká ako veľkosť bloku, označia sa byty ako alokované a overí sa či začiatok neukazoval na tento blok, ak hej nastaví sa na nasledujúci blok
  - ak zostávajúca veľkosť nového bloku nie je dostatočujúca pre vytvorenie nového voľného bloku, označia sa alokované byty a zvyšné ako visiace, overí sa začiatok ako v predchádzajúcom prípade
  - v poslednom prípade sa rozdelí blok. Metóda pridania nového bloku: **LIFO**. Označia sa alokované byty a vytvorí sa hlavička a pätička nového bloku. Overí sa či začiatok neukazoval na náš delený blok, ak hej nasledovník nového voľného bloku sa nastaví na nasledovníka deleného bloku. Inak nasledovník bude začiatočný blok. Previous nasledovníka ako aj začiatok sa nastaví na nový voľný blok.
  - na záver sa zamaskuje veľkosť alokovaného bloku a vráti pointer na úložisko tohto bloku

## ✓ MEMORY FREE

- **časová zložitosť:**  $O(n + m)$  - počet voľných blokov, memset
- vo väčšine prípadov je časová zložitosť  $O(n)$  – memset
- časová zložitosť  $O(n+m)$  nastane iba v prípade, že spájame 3 bloky, kedy musíme jeden z blokov odpojiť zo zoznamu

- **pamäťová zložitosť:**  $O(1)$

- **Postup:**

- prečítanie možných visiacich bytov pred hlavičkou a za pätičkou
- zistenie predchádzajúcej pätičky a nasledujúcej hlavičky
- ak je blok za a pred našim uvoľňovaným blokom alokovaný, veľkosť sa odmaskuje a uloží do dočasnej premennej. Hlavička a pätička sa posunie o možné visiace byty a overí sa či alokovaná veľkosť nie je menšia ako veľkosť potrebná na uchovanie ukazovateľov. Ak je menšia nastaví sa byty ako visiace a ukončí sa funkcia. V opačnom prípade sa uvoľnený blok pridá metódou **LIFO** ako bolo vysvetľované pri delení bloku v memory alloc, označia sa voľné byty a vráti 0.
- ak je blok pred našim uvoľňovaným blokom voľný a blok za alokovaný, nastáva spojenie týchto blokov. Veľkosť sa odmaskuje, zistí sa predchádzajúca hlavička, nastaví sa nová veľkosť pripočítajú sa taktiež možné visiace byty za uvoľňovaným blokom. Nastaví sa veľkosť v pätičke, označia voľné byty a vráti 0.
- ak je blok pred našim uvoľňovaným alokovaný a blok za voľný, nastáva spojenie týchto blokov. Hlavička sa posunie o možné visiace byty pred uvoľňovaným blokom, odmaskuje sa veľkosť, upraví sa nová veľkosť a pripočítajú možné visiace byty pred uvoľňovaným blokom. Ukazovateľ na predchádzajúci a nasledujúci blok sa nastaví na ukazovatele voľného bloku za uvoľňovaným a prepoja sa s novým blokom. Overí sa či začiatok neukazoval na voľný blok za, ak áno nastaví sa na novú hlavičku. Nastaví sa pätička, označia sa voľné byty a vráti 0.
- ak je blok za našim uvoľňovaným voľný a blok pred taktiež, nastáva spájanie týchto blokov. Zistí sa hlavička predchádzajúceho bloku, odmaskuje sa veľkosť a nová veľkosť sa upraví. Teraz musíme riešiť 3 možné prípady:
  - ak blok pred uvoľňovaným blokom ukazoval na blok za uvoľňovaným, nasledovník nášho nového bloku sa nastaví na nasledovníka bloku za uvoľňovaným blokom a ten sa prepojí s novým blokom.
  - ak nasledovník bloku za uvoľňovaným bol blok pred uvoľňovaným blokom, predchodca nového bloku sa nastaví na predchodcu bloku za uvoľňovaným a prepojí sa s novým blokom. Overí sa či začiatok

neukazoval na blok za uvoľňovaným, ak hej nastaví sa na nový spojený blok.

- v poslednom prípade, ak bloky pred a za nie sú nijak prepojené, overí sa či blok za uvoľňovaným mal predchodcu. Ak mal predchodcu musíme tento blok odpojiť zo zoznamu. Začneme prehľadávať celý zoznam, pokým nenatrafíme na blok, ktorý ukazuje na tento blok. Nastavíme nasledovníka tohto bloku na nasledovníka bloku za uvoľňovaným a prepojíme ho s týmto blokom. Ak blok za uvoľňovaným nemal predchodcu, znamená to, že tento blok bol začiatkový, preto sa začiatok nastaví na nasledovníka tohto bloku a jeho predchádzajúci sa nastaví na NULL.
- na koniec sa nastaví hlavička, označia voľné byty a vráti 0
- ak sa z nejakého dôvodu nepodarilo uvoľniť blok funkcia vráti 1

## ✓ **MEMORY CHECK**

- časová zložitosť =  $O(1)$
- pamäťová zložitosť =  $O(1)$

### ▪ **Postup:**

- zistí sa začiatok a koniec pamäte, hlavička a pätička
- overia sa podmienky za ktorých je ukazovateľ neplatný:
- ak ukazovateľ ukazuje za koniec alebo pred začiatok pamäte
- ak pätička ukazuje za koniec pamäte alebo hlavička pred začiatok pamäte
- ak veľkosť v hlavičke nie je nepárne číslo (alokovaný blok) alebo násobok 4
- ak sa veľkosť v hlavičke a pätičke nerovnajú
- v týchto prípadoch funkcia vráti 0
- v ostatných prípadoch je ukazovateľ platný a funkcia vráti 1



## TESTOVANIE

### Testovanie zo zadania

- testovanie zo zadania som implementoval vo vlastnej funkcii memory tester, ktorej vstupnými parametrami sú:
- minimálna a maximálna veľkosť bloku
- minimálna a maximálna veľkosť pamäte
- počet testov ak náhodne alokujeme a uvoľňujeme bloky (ak iba alokujeme musí byť počet testov 0 – alokujeme dokým stačí pamäť)
- premenná same\_sizes\_only, ktorú ak aktivujeme bude tester alokovať bloky rovnakej veľkosti
- premenná alloc\_only, ktorú ak aktivujeme bude prebiehať iba alokácia

#### ▪ **Postup:**

- funkcia náhodne zvolí veľkosť pamäte
- prebehne memory init
- na základe aktivovaných premenných bude prebiehať iba samotná alokácia alebo aj uvoľňovanie, či už rovnakých alebo rôznych veľkostí blokov
- ak iba alokujeme, tak dovtedy, kým máme dostatok pamäte
- ak aj uvoľňujeme, na základe počtu testov sa náhodne budú volať funkcie memory alloc alebo memory free
- jednotlivé ukazovatele, ktoré vracia memory alloc budeme ukladať do poľa, ako aj alokované veľkosti
- po prebehnutí testov funkcia odtestuje memory check pre všetky ukazovatele do našej pamäte, v prípade chybného memory checku vypíše správu o chybe
- na záver funkcia taktiež vypíše počet úspešne alokovaných blokov a bytov v percentuálnom vyjadrení

▪ **Test č. 1**

- veľkosť blokov: 8 – 24
- veľkosť pamäte: 50 – 200

```
*Allocating blocks of size: 8*  
Memory size: 160  
Allocated blocks: 40.00%  
Allocated bytes: 40.00%
```

```
*Allocating and freeing blocks of size: 10*  
Memory size: 70  
Allocated blocks: 61.11%  
Allocated bytes: 61.11%
```

▪ **Test č. 2**

- veľkosť blokov: 8 – 24
- veľkosť pamäte: 50 – 200

```
*Allocating blocks of random size*  
Memory size: 126  
Allocated blocks: 50.00%  
Allocated bytes: 56.52%
```

```
*Allocating and freeing blocks of random size*  
Memory size: 51  
Allocated blocks: 57.14%  
Allocated bytes: 59.63%
```

▪ **Test č. 3**

- veľkosť blokov: 500 – 5000
- veľkosť pamäte: 10000 – 100000

```
*Allocating blocks of random size*  
Memory size: 34254  
Allocated blocks: 100.00%  
Allocated bytes: 100.00%
```

```
*Allocating and freeing blocks of random size*  
Memory size: 15788  
Allocated blocks: 92.31%  
Allocated bytes: 86.20%
```

▪ **Test č. 4 – alokácia**

- veľkosť blokov: 8 – 50000
- veľkosť pamäte: 100000 – 1000000

```
*Allocating blocks of random size*  
Memory size: 114201  
Allocated blocks: 100.00%  
Allocated bytes: 100.00%
```

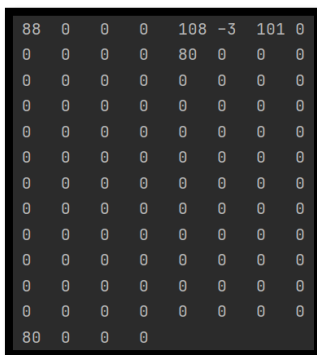
```
*Allocating and freeing blocks of random size*  
Memory size: 106331  
Allocated blocks: 94.78%  
Allocated bytes: 89.21%
```

## Vlastné testy

- tieto testy možno nájsť zakomentované pod funkciou main
- cieľ týchto testov je overiť hraničné situácie, ktoré môžu nastať najmä pri uvoľňovaní blokov, ale aj samotnej alokácii
- nižšie uvedené obrázky vizuálne reprezentujú našu pamäť, kde môžeme vidieť zmenu hlavičiek, pätičiek ako aj ukazovateľov
- reprezentácia bytov:
- 0 = voľný byte, 2 = alokovaný byte, -2 = visiacy byte

### ▪ **Test č. 1 - memory init**

- veľkosť pamäte = 100

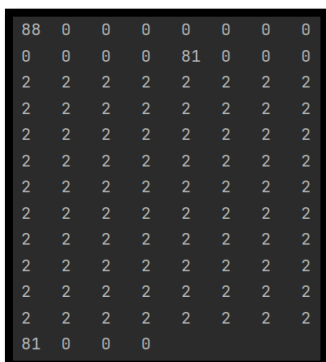


A 16x8 grid representing memory layout. The first row contains values: 88, 0, 0, 0, 108, -3, 101, 0. The second row contains: 0, 0, 0, 0, 80, 0, 0, 0. The third row contains: 0, 0, 0, 0, 0, 0, 0, 0. The fourth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The fifth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The sixth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The seventh row contains: 0, 0, 0, 0, 0, 0, 0, 0. The eighth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The ninth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The tenth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The eleventh row contains: 0, 0, 0, 0, 0, 0, 0, 0. The twelfth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The thirteenth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The fourteenth row contains: 0, 0, 0, 0, 0, 0, 0, 0. The fifteenth row contains: 80, 0, 0, 0, 0, 0, 0, 0.

- následné memory alloc testy boli vykonané po memory init testu vyššie

### ▪ **Test č. 2 – memory alloc**

- veľkosť pamäte = 100, alokovaná veľkosť = 80
- alokovanie bloku, ktorého veľkosť je rovnaká ako veľkosť voľného bloku



A 16x8 grid representing memory layout. The first row contains values: 88, 0, 0, 0, 0, 0, 0, 0. The second row contains: 0, 0, 0, 0, 81, 0, 0, 0. The third row contains: 2, 2, 2, 2, 2, 2, 2, 2. The fourth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The fifth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The sixth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The seventh row contains: 2, 2, 2, 2, 2, 2, 2, 2. The eighth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The ninth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The tenth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The eleventh row contains: 2, 2, 2, 2, 2, 2, 2, 2. The twelfth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The thirteenth row contains: 2, 2, 2, 2, 2, 2, 2, 2. The fourteenth row contains: 81, 0, 0, 0, 0, 0, 0, 0.

▪ **Test č. 3 – memory alloc**

- veľkosť pamäte = 100, alokovaná veľkosť = 60
- alokovanie bloku, ktorého veľkosť po rozdelení nie je postačujúca na vytvorenie nového bloku

88	0	0	0	0	0	0	0
0	0	0	0	61	0	0	0
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	61	0	0	0
-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2				

▪ **Test č. 4 – memory alloc**

- veľkosť pamäte = 100, alokovaná veľkosť = 40
- alokovanie bloku a jeho následné rozdelenie

88	0	0	0	-100	-3	101	0
0	0	0	0	41	0	0	0
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
41	0	0	0	32	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
32	0	0	0				

▪ **Test č. 5 – memory free**

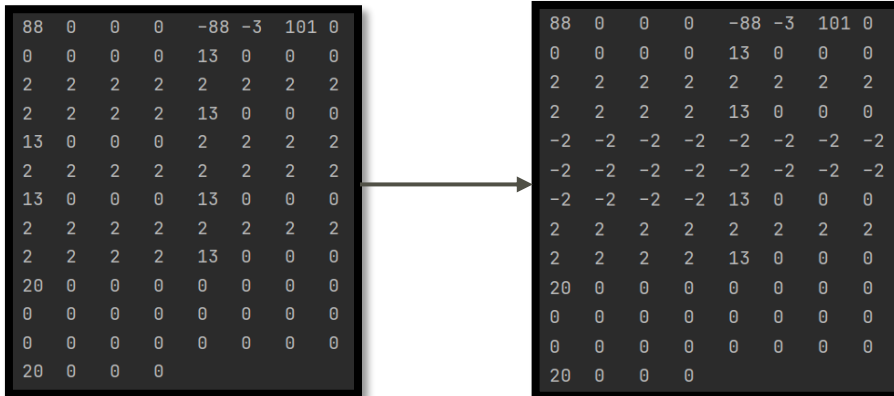
- uvoľňovanie bloku, kedy blok pred ním aj za ním je alokovaný

88	0	0	0	0	0	0	0
0	0	0	0	13	0	0	0
2	2	2	2	2	2	2	2
2	2	2	2	13	0	0	0
33	0	0	0	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	33	0	0	0
13	0	0	0	2	2	2	2
2	2	2	2	2	2	2	2
13	0	0	0	-2	-2	-2	-2
-2	-2	-2	-2				

88	0	0	0	-128	-3	101	0
0	0	0	0	13	0	0	0
2	2	2	2	2	2	2	2
2	2	2	2	13	0	0	0
32	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	32	0	0	0
13	0	0	0	2	2	2	2
2	2	2	2	2	2	2	2
13	0	0	0	-2	-2	-2	-2
-2	-2	-2	-2				

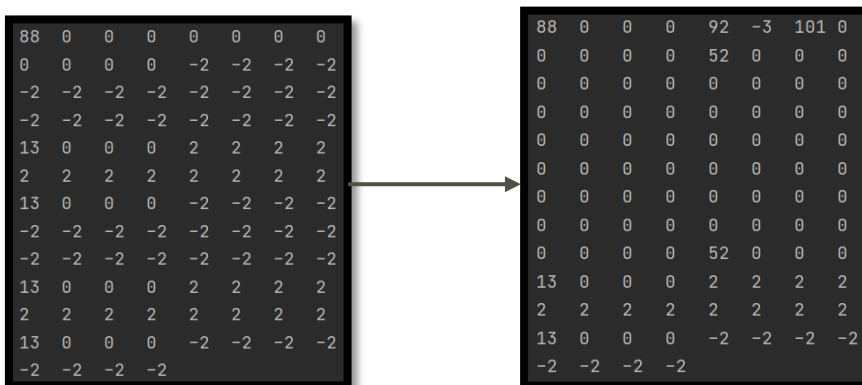
▪ **Test č. 6 – memory free**

- uvoľňovanie bloku, kedy blok pred ním aj za ním je alokovaný a veľkosť uvoľneného bloku nestačí na vytvorenie nového bloku



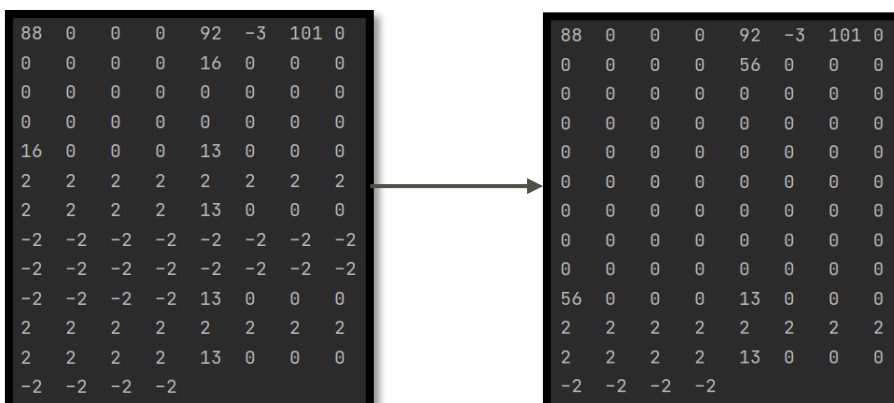
▪ **Test č. 7 – memory free**

- uvoľňovanie bloku, kedy visia byty pred aj za blokom



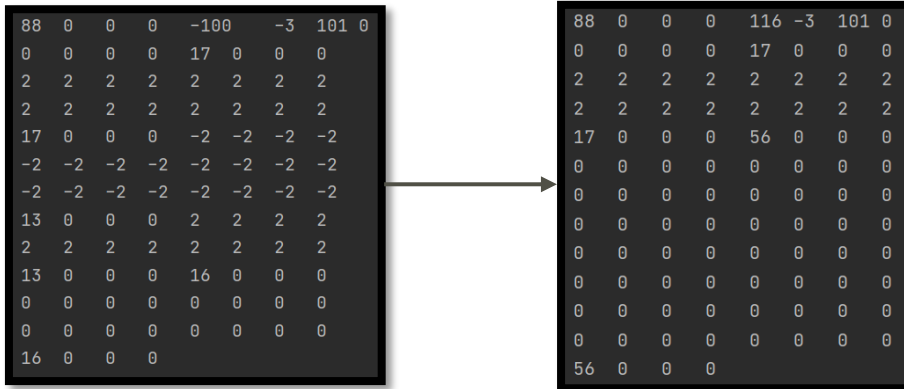
▪ **Test č. 8 – memory free**

- uvoľňovanie bloku, kedy blok pred ním je voľný, blok za ním je alokovaný a visia byty za blokom



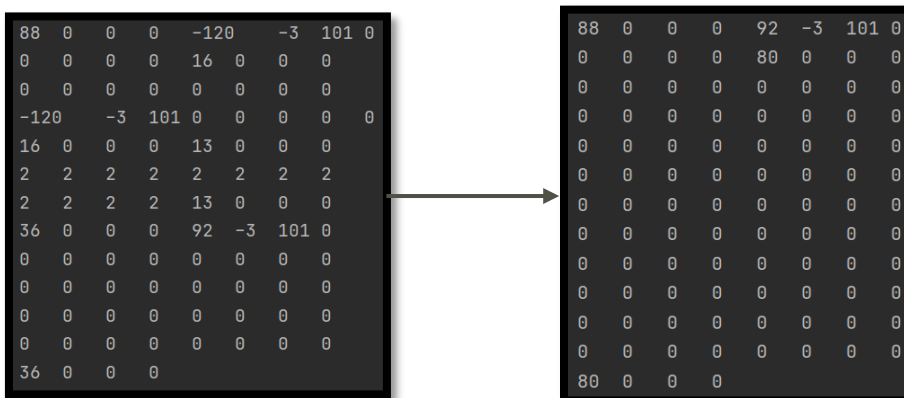
▪ **Test č. 9 – memory free**

- uvoľňovanie bloku, kedy blok pred ním je alokovaný, blok za ním je voľný a visia byty pred blokom



▪ **Test č. 10 – memory free**

- uvoľňovanie bloku, kedy blok pred ním aj za ním je voľný



• Testovanie efektívnosti

- môj prípad / ideálny prípad
- veľkosť pamäte: 1000
- alokácia:
- 1 bytové bloky: 8,1 %
- 4 bytové bloky: 32,4 %
- 8 bytové bloky: 48,8 %
- 20 bytové bloky: 70 %
- 50 bytové bloky: 80 %

## ZÁVER

Snažil som sa vypracovať metódu, ktorá bude časovo efektívna najmä pri použití väčších blokov pamäte a alokácii väčších blokov. V dôsledku použitia `unsigned int` má užívateľ možnosť alokovať bloky naozaj veľkej veľkosti. V prípade menších blokov pamäte som sa snažil maximalizovať efektívnosť tým, že nezvyšujem ich veľkosť, aj keď už z nich nebude možno vytvoriť následne voľný blok, ale byty po uvoľnení zostanú visieť. Používateľ má takto možnosť alokovať viacero blokov malej veľkosti. Premazávaním pamäte sa táto metóda skôr podobá na funkciu `calloc`.