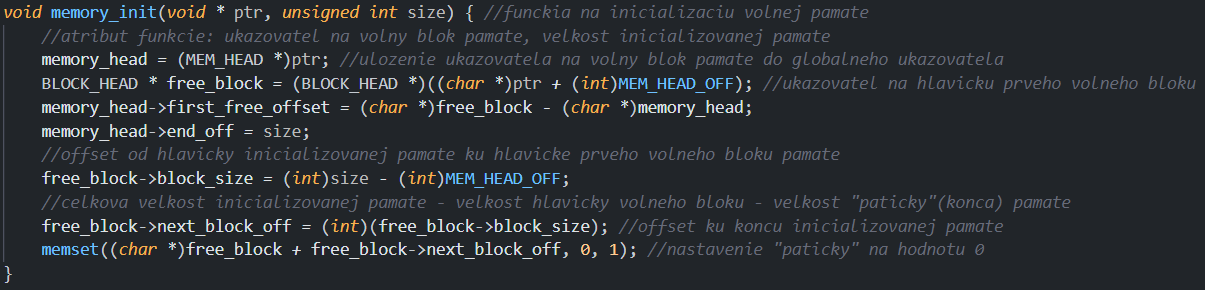
[DSA] Zadanie 1 – Správca pamäti

Autor: Ema Richnáková

Pre dané riešenie som si vybrala explicitný zoznam bez zoznamov blokov voľnej pamäti.

4 hlavné funkcie programu

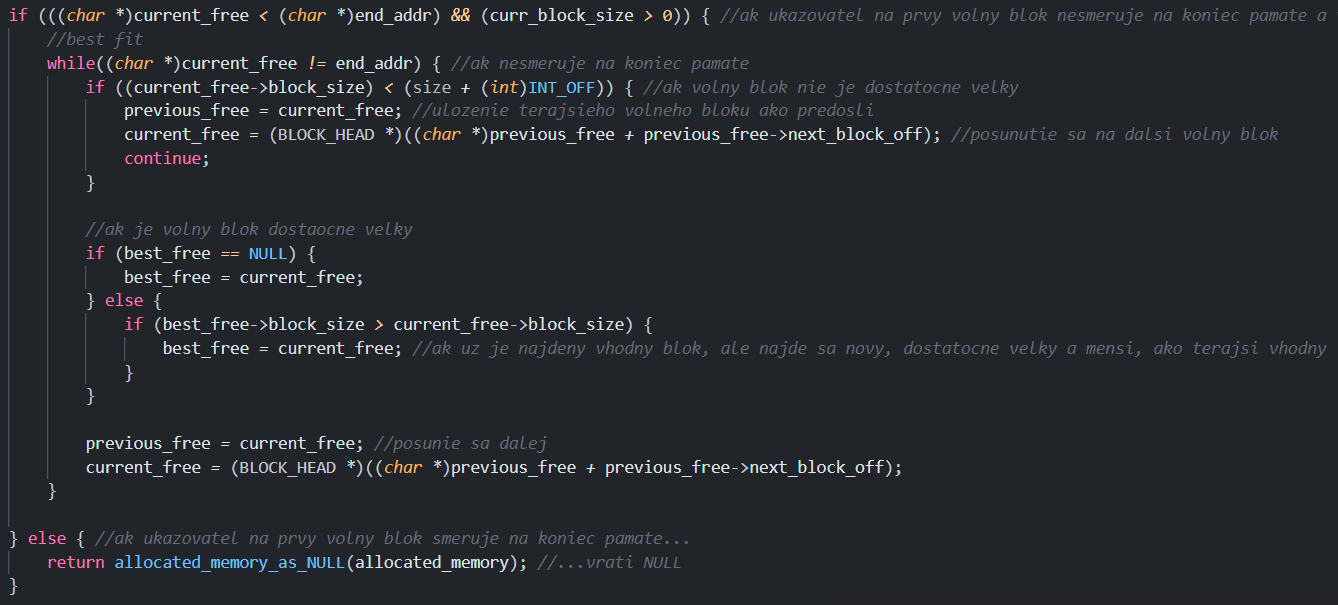
**void memory\_init(void \*ptr, unsigned int size);**  
  
Funkcia **memory\_init** slúži na inicializáciu spravovanej pamäte. Funkcia sa volá len raz a to vo funkcii *int\_main* pred všetkými volaniami iných funkcii*.* Vstupné parametre sú: ukazovateľ na blok pamäte, ktorá bude slúžiť na organizovanie a aj prideľovanie voľnej pamäte a veľkosť daného bloku.



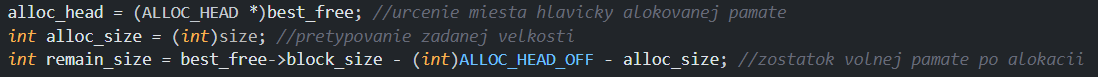
* Do globálnej premennej \**memory\_head* sa ukladá adresa, kde začína inicializovaný blok pamäti. \**memory\_head* je zároveň hlavička celého bloku pamäte, v ktorej sa ukladá offset (počet bajtov) k ďalšiemu voľnému bloku pamäte a offset ku koncu pamäti., čo hneď na začiatku predstavuje zabratie celkovej pamäte 8B.

Ukazovateľ \**free\_block* je hlavička prvého voľného bloku v pamäti, ktorá sa nachádza hneď za hlavičkou \**memory\_head* a uschováva v sebe 2 údaje, t.j. veľkosť voľného bloku a offset na ďalší voľný blok.

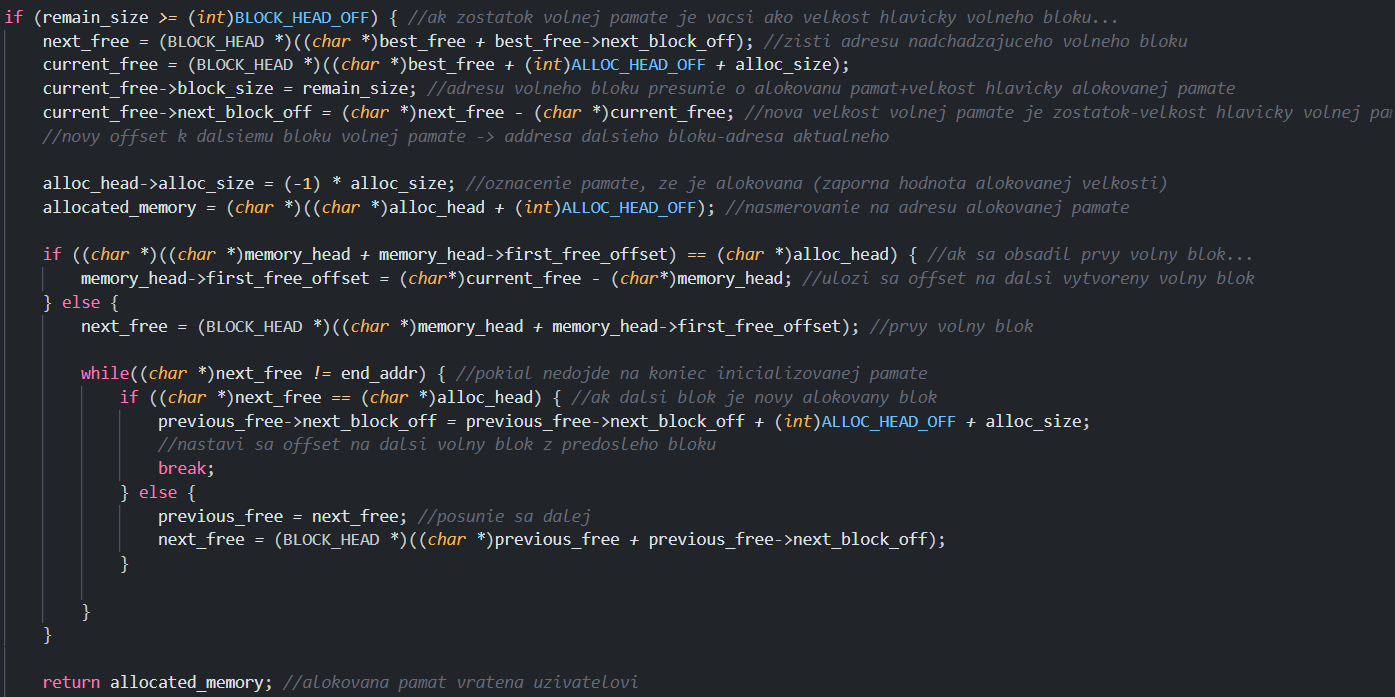
**void \*memory\_alloc(unsigned int size);**  
  
Funkcia **memory\_alloc** vykonáva rovnakú službu ako štandardný *malloc*. Vstupný parameter je veľkosť požadovaného súvislého bloku pamäte a funckia vráti ukazovateľ na úspešne alokovaný blok voľnej pamäte, ktorý vyhradil, alebo *NULL*, ak nebolo vyhradenie vhodného bloku pamäte možné.



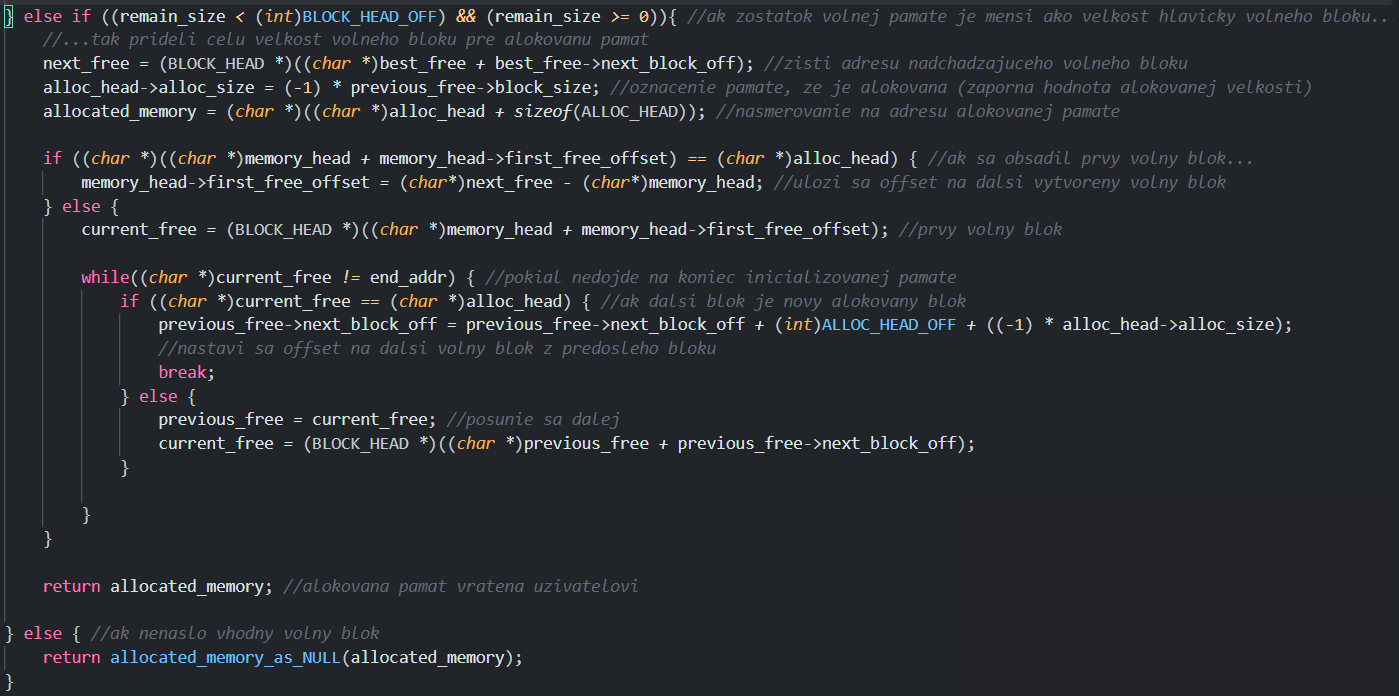
* Prvá signifikantná časť funkcie je zistenie, či je stále dostupná voľná pamäť. Ak nie, funkcia vráti *NULL,* a ak je dostupná voľná pamäť, zisťuje, či aktuálny blok pamäte je dostatočne veľký pre požadovaný blok alokovanej pamäte. Algoritmus hľadania – Best Fit – prejde všetky voľné bloky a vyberie z nich najvhodnejší, t.j. najmenší, aký mohol nájsť v pamäti, no dostatočne veľký, aby sa doň zmestila veľkosť alokovanej pamäte + hlavička alokovanej pamäte (ktorá má 4B).



* Ak nájde vhodný voľný blok, na začiatok nastaví hlavičku alokovaného bloku a určí, koľko miesta z voľného bloku zostane po alokácii.



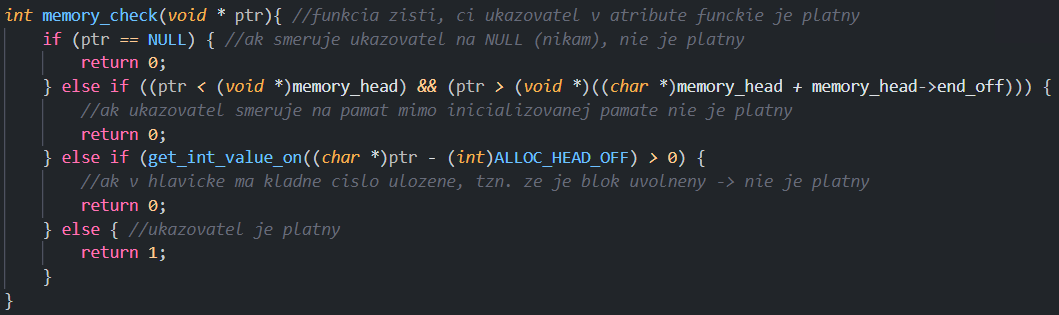
* Ak zostatkové miesto je dostatočne veľké aspoň na uloženie hlavičky nového voľného bloku, tak alokovanému bloku pridelí požadovanú veľkosť (zadanú od používateľa) a zvyšok veľkosti priradí novému (menšiemu) voľnému bloku pamäti a keďže sa hlavička voľného bloku posunula v pamäti, musí sa upraviť aj offset k nasledujúcemu voľnému bloku, a ak alokovaný blok bol ako prvý v pamäti, tak aj offset v hlavičke celkovej pamäte musí nastaviť na posunutú hodnotu voľného bloku. A ak pred alokovaným blokom je voľný blok, taktiež treba posunúť offset.  
  V hlavičke alokovaného bloku sa uloží záporná hodnota jeho veľkosti, aby bolo jasné, že daný blok pamäte je už alokovaný.



* Ak zostatkové miesto nie je dostatočne veľké aspoň na uloženie hlavičky voľného bloku (t.j.8B), tak alokovanému bloku pridelí veľkosť celého voľného bloku (keďže by mohlo dochádzať k prekrývaniu jednotlivých blokov). V hlavičke alokovaného bloku sa uloží záporná hodnota jeho veľkosti, aby bolo jasné, že daný blok pamäte je už alokovaný.   
  V premennej *next\_free* je uložená adresa na ďalší voľný blok a k nej treba nastaviť offset buď z celkovej hlavičky pamäte alebo z predošlého voľného bloku, ak nejaký existuje.

Inak ak funkcia nenájde vhodný voľný blok, vracia *NULL.*

**int memory\_check(void \*ptr);**  
  
Funkcia **memory\_check** slúži na kontrolu parametra *\*ptr*, či daný smerník je platný, tzn. ak bol v nejakom predchádzajúcom volaní vrátení funkciou *memory\_alloc* a nebol uvoľnený funkciou *memory\_free.* Funkcia vráti 1, ak je platný a 0, ak je neplatný.



* Ak ukazovateľ *\*ptr* je NULL alebo ukazuje na adresu pred celkový inicializovaný blok alebo za celkový inicializovaný blok alebo je na danej adrese kladné číslo (znak voľného bloku), je neplatný.

**int memory\_free(void \*valid\_ptr);**  
  
Funkcia **memory\_free** slúži na uvoľnenie vyhradeného bloku pamäti. Spĺňa rovnakú funkciu ako štandartná funkcia *free*. Funkcia vráti 0, ak sa úspešne uvoľnila pamäť, inak vráti 1.

Najprv je argument funkcie skontrolovaný, či je ukazovateľ platný. Ak nie je, funkcia *free* vracia 1, inak sa vykoná telo funkcie.

Telo funkcie je rozdelené na 2 prípady:

1. ak uvoľňovaná pamäť je pred prvým voľným blokom

2. ak uvoľňovaná pamäť je za prvým voľným blokom

V 1. prípade stačí zistiť, či za uvoľňovanou pamäťou je fyzicky v pamäti ďalší voľný blok.

1. Ak áno, uvoľňovanú pamäť zlúči s voľným blokom a upraví offset v hlavičke k celkovej pamäti, aby odkazoval na adresu uvoľnenej pamäti. A ak predošlý voľný blok odkazoval offsetom na ďalší voľný blok, treba upraviť offset uvoľnenej pamäte, aby ukazoval na ten ďalší voľný blok.
2. Ak nie, uvoľní pamäť a upraví offset v hlavičke k celkovej pamäti, aby odkazoval na adresu uvoľnenej pamäti. A ak za uvoľňovanou pamäťou sa nachádza voľný blok (nie fyzicky), tak offset v uvoľnenej pamäti nastaví na ďalší voľný blok.

V 2. prípade sa snaží nájsť voľnú pamäť čo najbližšie (fyzicky) pred uvoľňovanou pamäťou. Ak koniec predošlej voľnej pamäte je presne pred začiatkom uvoľňovanej, zlúči ich dokopy a vznikne nová voľná pamäť. A ak predošlá voľná pamäť nespĺňa podmienku, tak sa uvoľňovaná pamäť uvoľní a v hlavičke predošlej pamäte sa zmení offset, aby odkazoval na uvoľnenú pamäť.   
A ak by za uvoľňovanou pamäťou bol fyzicky voľný blok, tak postupujeme ako v 1. prípade.  
Postup v 2. prípade je taký, že najprv zlúči voľný blok za uvoľňovanou pamäťou (ak je to možné) a až potom zlučuje predošlý blok (znova, ak je to možné).

Testovanie

Program som testovala 4 rôznymi formami:

1) Teoretická forma (kreslenie na papier)

2) Vizuálna forma v programe (vypisovanie hodnôt)

3) Formou simulovania scenárov zo zadania 1

4) Testovanie spájania blokov

1) Teoretická forma (kreslenie na papier)

Pred samotným písaním kódu mi pomohli prvú štruktúru navrhnúť pero a papier. Pri kreslení rôznych scenárov som si rozvrhla podmienky v jednotlivých funkciách a častiach funkcii.

2) Vizuálna forma v programe

Pomocou jednoduchej funkcie *printf* som vedela presne kontrolovať hodnoty jednotlivých premenných.

3) Formou simulovania rôznych scenárov zo zadania 1

Testovanie daných scenárov je prevádzané pomocou funkcií, ktoré sú uvedené v súbore *testovanie\_z1.c*.

Funkcie:

1. test1();
   * 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)
2. test2();
   * 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)
3. test3();
   * prideľovanie nerovnakých blokov väčšej veľkosti (veľkosti 500 až 5000 bytov) pri použití väčších celkových blokov pre správcu pamäte (aspoň veľkosti 1000 bytov)
4. test4();
   * 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)

Výpis z testov (pri generovaní náhodných veľkostí na alokáciu):

Size of MEM\_HEAD: 8

Size of BLOCK\_HEAD: 8

Size of ALLOC\_HEAD: 4

BEST FIT WITHOUT DYNAMIC HEADS

test1

-----------------------------------test same free block

pri pouziti malych celkoych blokov pre spravcu pamate (do 50 bytov, do 100 bytov, do 200 bytov)

-----MALLOC TEST ENDED-----

Itterations: 1 -> koľkokrát alokovalo pamäť

Alloc block size: 23 -> veľkosť alokovaného bloku

Taken by user: 23/50 -> 46.00% ->koľko z pamäte využíva používateľ

Taken memory: 35/50 -> 70.00% ->koľko z pamäte je zabraných celkovo (započítaná alokovaná pamäť aj všetky hlavičky)

Taken usage: 23/35 -> 65.71% ->koľko zo zabratej pamäte využíva použivateľ

-----FREE TEST ENDED----- -> pri chybnom uvoľňovaný by vypísalo "ERROR"

-----MALLOC TEST ENDED-----

Itterations: 4

Alloc block size: 15

Taken by user: 60/100 -> 60.00%

Taken memory: 84/100 -> 84.00%

Taken usage: 60/84 -> 71.43%

-----FREE TEST ENDED-----

-----MALLOC TEST ENDED-----

Itterations: 8

Alloc block size: 19

Taken by user: 152/200 -> 76.00%

Taken memory: 192/200 -> 96.00%

Taken usage: 152/192 -> 79.17%

-----FREE TEST ENDED-----

test2

-----------------------------------test different free block

pri pouziti malych celkoych blokov pre spravcu pamate (do 50 bytov, do 100 bytov, do 200 bytov)

22

-----MALLOC TEST ENDED-----

Itterations: 1

Alloc block size: 0 -> 0 je tu z toho dôvodu, že alokované veľkosti sa môžu líšiť

Taken by user: 22/50 -> 44.00%

Taken memory: 34/50 -> 68.00%

Taken usage: 22/34 -> 64.71%

-----FREE TEST ENDED-----

10 19 19 24 -> vypísanie jednotlivých veľkostí alokovanej pamäte

-----MALLOC TEST ENDED-----

Itterations: 4

Alloc block size: 0

Taken by user: 72/100 -> 72.00%

Taken memory: 96/100 -> 96.00%

Taken usage: 72/96 -> 75.00%

-----FREE TEST ENDED-----

24 19 9 12 22 11

18 9 21

-----MALLOC TEST ENDED-----

Itterations: 9

Alloc block size: 0

Taken by user: 145/200 -> 72.50%

Taken memory: 189/200 -> 94.50%

Taken usage: 145/189 -> 76.72%

-----FREE TEST ENDED-----

test3

-----------------------------------test different free block

pri pouziti vacsich celkovych blokov pre spravcu pamate (aspon velkosti 1000 bytov)

894

-----MALLOC TEST ENDED-----

Itterations: 1

Alloc block size: 0

Taken by user: 894/1000 -> 89.40%

Taken memory: 906/1000 -> 90.60%

Taken usage: 894/906 -> 98.68%

-----FREE TEST ENDED-----

582 3368

-----MALLOC TEST ENDED-----

Itterations: 2

Alloc block size: 0

Taken by user: 3950/5000 -> 79.00%

Taken memory: 3966/5000 -> 79.32%

Taken usage: 3950/3966 -> 99.60%

-----FREE TEST ENDED-----

4305 2981

-----MALLOC TEST ENDED-----

Itterations: 2

Alloc block size: 0

Taken by user: 7286/10000 -> 72.86%

Taken memory: 7302/10000 -> 73.02%

Taken usage: 7286/7302 -> 99.78%

-----FREE TEST ENDED-----

test4

-----------------------------------test different free block

pri pouziti vacsich celkovych blokov pre spravcu pamate (aspon velkosti 1000 bytov)

825

-----MALLOC TEST ENDED-----

Itterations: 1

Alloc block size: 0

Taken by user: 825/1000 -> 82.50%

Taken memory: 837/1000 -> 83.70%

Taken usage: 825/837 -> 98.57%

-----FREE TEST ENDED-----

3211 713

-----MALLOC TEST ENDED-----

Itterations: 2

Alloc block size: 0

Taken by user: 3924/5000 -> 78.48%

Taken memory: 3940/5000 -> 78.80%

Taken usage: 3924/3940 -> 99.59%

-----FREE TEST ENDED-----

5058 415

-----MALLOC TEST ENDED-----

Itterations: 2

Alloc block size: 0

Taken by user: 5473/10000 -> 54.73%

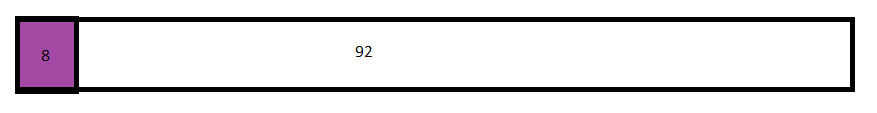
Taken memory: 5489/10000 -> 54.89%

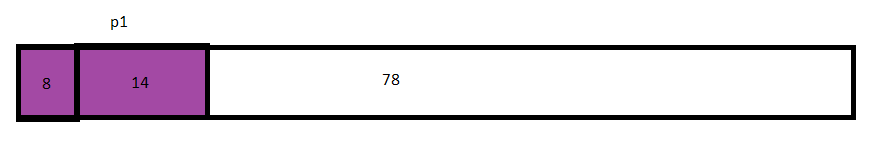
Taken usage: 5473/5489 -> 99.71%

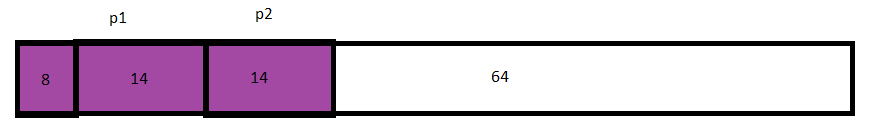
-----FREE TEST ENDED-----

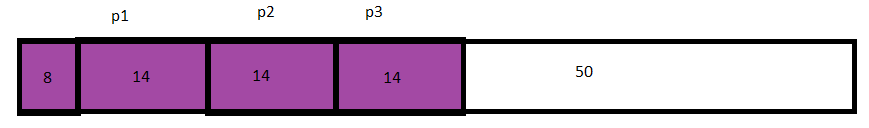
4) Testovanie spájania blokov

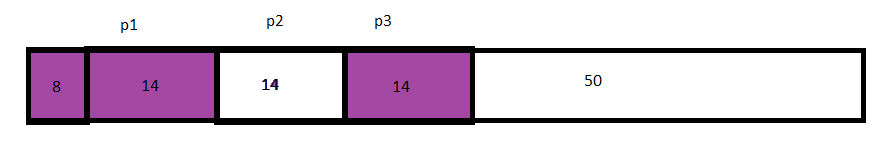
* priebeh testu vizuálne:

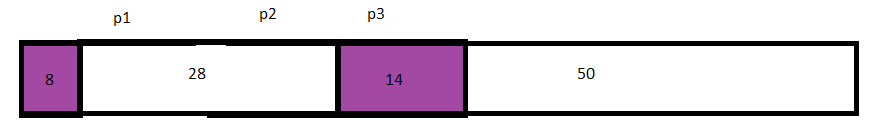


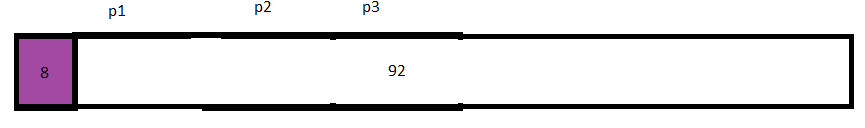












Odhad zložitostí

**Odhad časovej zložitosti je O(*n\*k*)**, kde *n* predstavuje počet vykonaných operácii a *k*počet voľných blokov.

**Odhad priestorovej zložitosti je O(*m\*l + o\*r*)**, kde *m* predstavuje veľkosť hlavičky voľného bloku, *l*počet voľných blokov*, o*veľkosť hlavičky zabratého bloku *a r* počet zabratých blokov.

Zhodnotenie

Výsledný program sa správa ako štandardné funkcie *malloc* alebo *free.*

Program efektívne využíva pamäť, aj keď táto efektivita by sa dal ešte navýšiť a to tak, keby celá štruktúra voľných blokov bola postavená na spájanom zozname voľných blokov usporiadaných podľa veľkosti, veľkosti hlavičiek by sa prispôsobovali veľkosti celkovej inicializovanej.