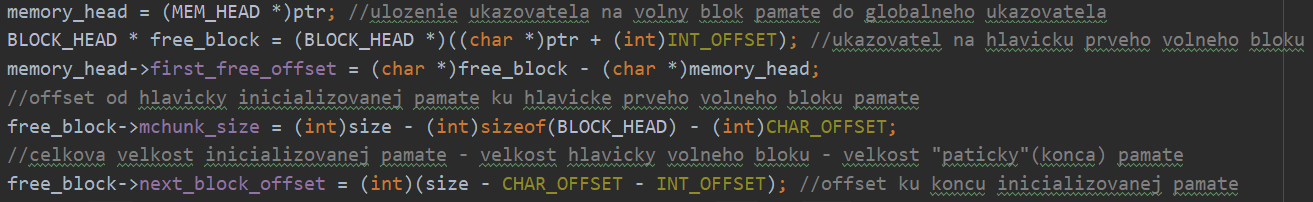
[DSA] Zadanie 1 – Správca pamäti

Autor: Ema Richnáková

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

4 hlavné funkcie programu

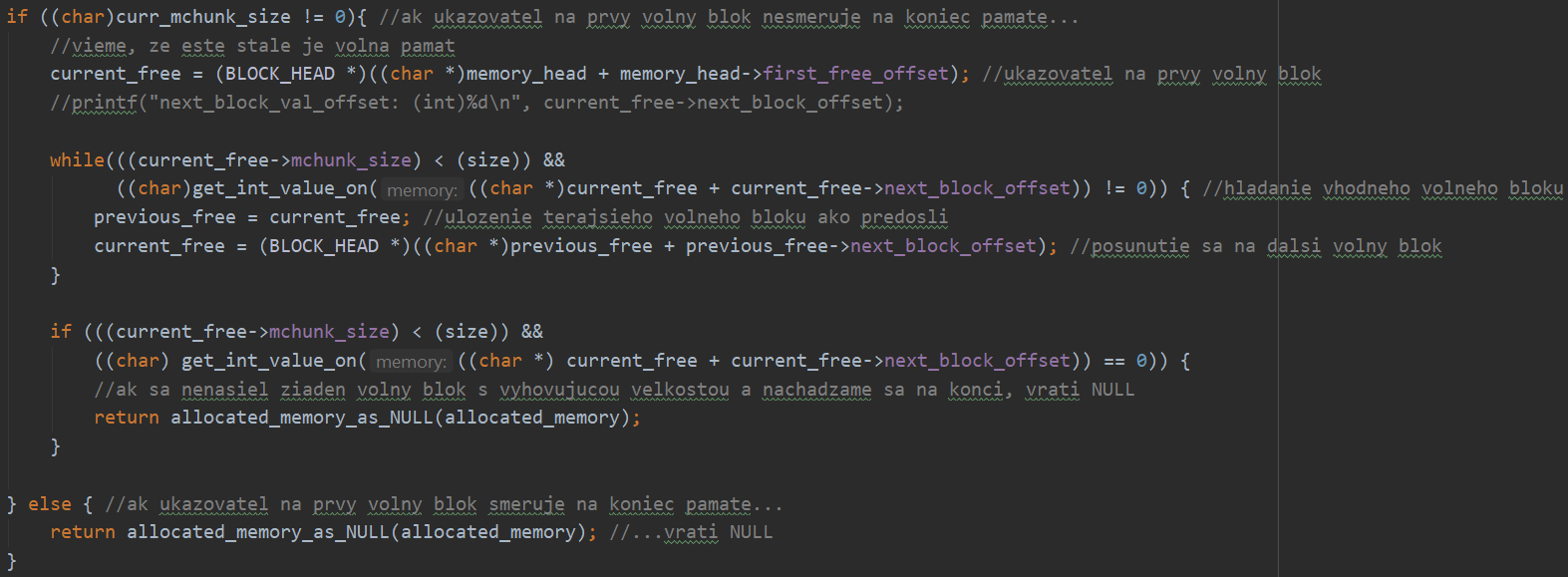
**void memory\_init(void \*ptr, unsigned int size);**  
  
Funkcia **memory\_init** slúži na inicializáciu spravovanej pamäte. Funkcia sa volá len raz a to vo funkcii *int\_main* pred všetkými volaniami iných funkcii*.* Vstupné parametre sú: ukazovateľ na blok pamäte, ktorá bude slúžiť na organizovanie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



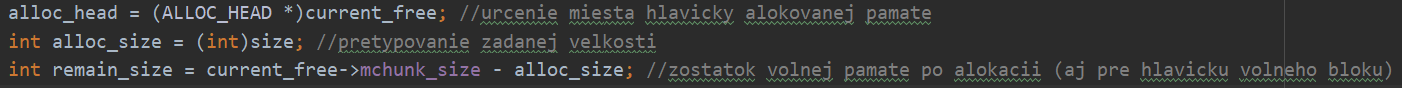
* Do globálnej premennej \**memory\_head* sa ukladá adresa, kde začína inicializovaný blok pamäti. \**memory\_head* je zároveň hlavička celého bloku pamäte, v ktorej sa ukladá offset (počet bajtov) k ďalšiemu voľnému bloku pamäte, čo hneď na začiatku predstavuje 4B.

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, ktorú môže používateľ v *memory\_alloc* využiť a offset na ďalší voľný blok.

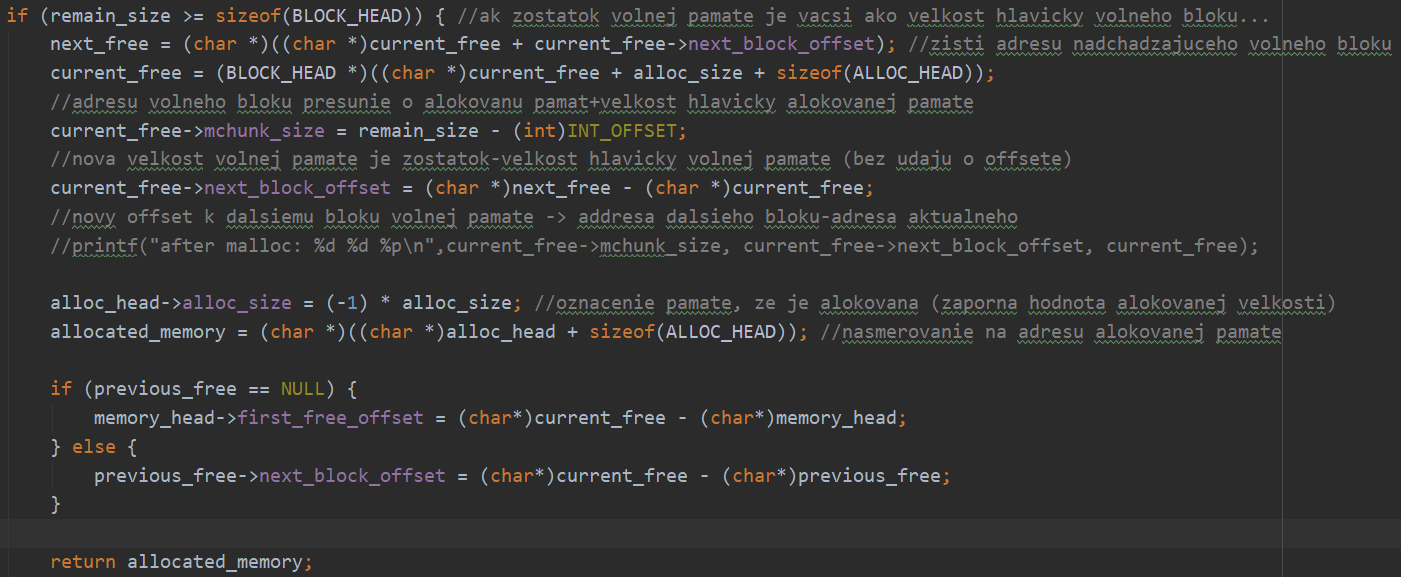
**void \*memory\_alloc(unsigned int size);**  
  
Funkcia **memory\_alloc** vykonáva rovnakú službu ako štandardný *malloc*. Vstupný parameter je veľkosť požadovaného súvislého bloku pamäte a funckia vráti ukazovateľ na úspešne alokovaný blok voľnej pamäte, ktorý vyhradil, alebo *NULL*, ak nebolo vyhradenie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



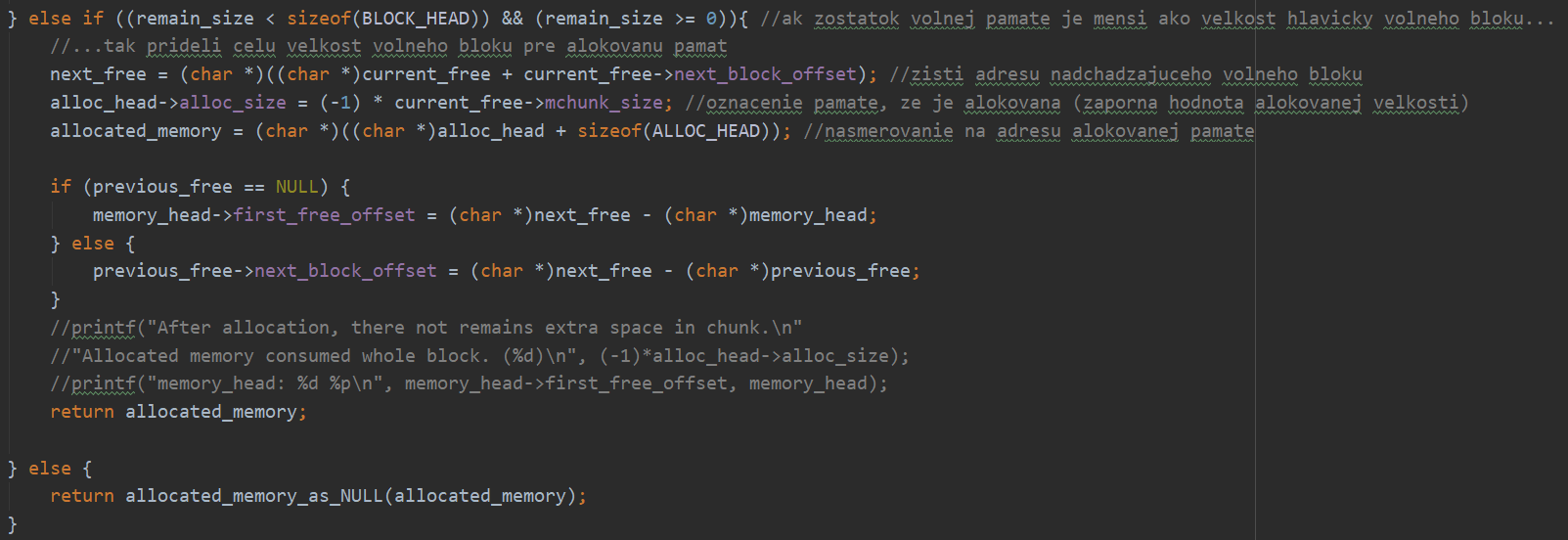
* Prvá signifikantná časť funkcie je zistenie, či je stále dostupná voľná pamäť. Ak nie, funkcia vráti *NULL,* a ak je dostupná voľná pamäť, zisťuje, či aktuálny blok pamäte je dostatočne veľký pre požadovaný blok alokovanej pamäte. Algoritmus hľadania – First Fit – prvý vhodný blok zaberie.



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



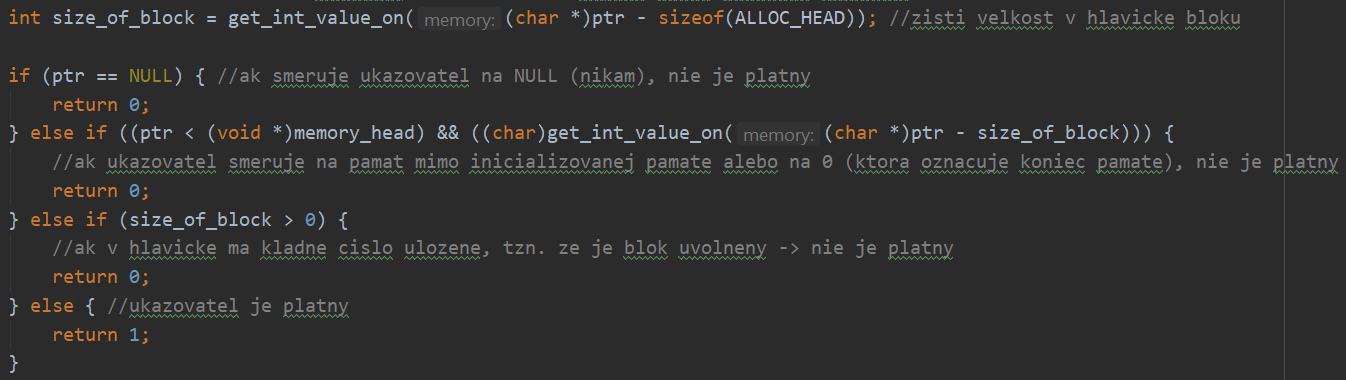
* Ak zostatkové miesto je dostatočne veľké aspoň na uloženie hlavičky voľného bloku, tak alokovanému bloku pridelí požadovanú veľkosť (zadanú od používateľa) a zvyšok veľkosti priradí novému (menšiemu) voľnému bloku pamäti a keďže sa hlavička voľného bloku posunula v pamäti, musí sa upraviť aj offset k ďalšiemu voľnému bloku a aj offset v hlavičke celkovej pamäte musí nastaviť na posunutú hodnotu voľného bloku.  
  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, 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.



* Premenná *size\_of\_block* si ukladá veľkosť bloku z jeho hlavičky.

Ak ukazovateľ *\*ptr* ukazuje na adresu pred celkový inicializovaný blok alebo ukazuje na 0 (koniec pamäte), 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

3) Formou simulovania rôznych scenárov

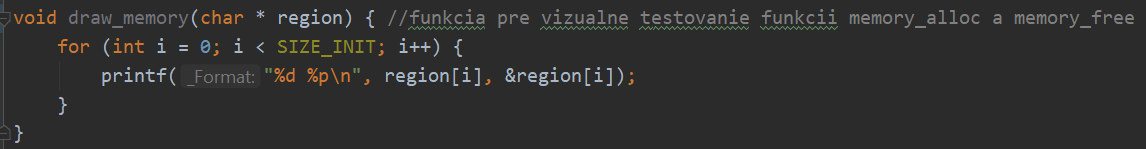
4) Forma ,,Pokus-omyl‘‘

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 *draw\_memory*, som pri menších veľkostiach pamäte vedela bajt po bajte kontrolovať, či sa funkcie správajú podľa očakávaní alebo nie.



* Funkcia *void draw\_memory(char \* region)*

3) Formou simulovania rôznych scenárov

Manuálne som zadala postupnosť volaní *memory\_alloc a memory\_free* a sledovala výsledky, vďaka výpisom z jednotlivých častí funkcii (v programe som dôležité výpisy nechala, ale sú zakomentované).

Najčastejšie scenáre:

* Postupná alokácia celej pamäte rovnakými blokmi a postupné uvoľnenie všetkých blokov
* Postupná alokácia celej pamäte rôznymi blokmi a postupné uvoľnenie všetkých blokov
* Uvoľňovanie blokov, ak je fyzicky za uvoľňovaným blokom voľný blok
* Uvoľňovanie blokov, ak nie je fyzicky za uvoľňovaným blokom voľný blok
* Uvoľňovanie blokov, ak je fyzicky pred uvoľňovaným blokom voľný blok
* Uvoľňovanie blokov, ak nie je fyzicky pred uvoľňovaným blokom voľný blok
* Uvoľňovanie blokov, ak je blok posledný v celkovej pamäti
* Uvoľňovanie blokov, ak je blok prvý v celkovej pamäti

4) Forma ,,Pokus-omyl‘‘

(Bohužial) Aj touto formou testovania som opravovala chyby a to boli hlavne chyby týkajúce sa pretypovávania premenných, kedy buď ľudské oko zlyhalo a prehliadlo danú chybu alebo dané pretypovanie mi neprišlo pri určitých situáciách potrebné.

**Dôležité: Dokumentáciu je potrebné uložiť stlačením Uložiť (Ctrl+S).**

Odhad zložitostí

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

**Odhad priestorovej zložitosti je O(*5 + m\*l + o\*r*)**, kde *m* predstavuje velkost hlavicky volneho bloku, *l pocet volnych blokov, o velkost hlavicky zabrateho bloku bloku a r pocet zabratych 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 pamäti a vo funkcii *alloc\_memory* bolo vyhľadanie vhodného voľného bloku podľa algoritmu *Best Fit* – kedy by našiel taký voľný blok, ktorý je rovnako veľký ako požadovaná veľkosť alebo by sa tejto veľkosti najviac približoval (ale pri tejto možnosti by mohla utrpieť časová efektivita vykonania programu).