Fakulta informatiky a informačných technológií STU v Bratislave Ilkovičova 2, 842 16 Bratislava 4

Tibor Dulovec

**Zadanie 1 – Správca pamäti**

DSA LS 2020/21  
Cvičiaci P. Lehoczký  
Pondelok 11:00 – 12:50

**Obsah**

[Opis programu 3](#_Toc65969677)

[Hlavička bloku 3](#_Toc65969678)

[Opis funkcií 3](#_Toc65969679)

[memory\_init 3](#_Toc65969680)

[memory\_alloc 3](#_Toc65969681)

[memory\_free 3](#_Toc65969682)

[memory\_check 4](#_Toc65969683)

[Testovanie programu 5](#_Toc65969684)

[Test1: Prideľovanie malých blokov rovnakej veľkosti 5](#_Toc65969685)

[Test2: prideľovanie blokov rôznej malej veľkosti 6](#_Toc65969686)

[Test3: pridávanie rôzny počet náhodne veľkých blokov 7](#_Toc65969687)

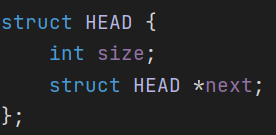
[Test4: Uvoľňovanie a spájanie blokov 8](#_Toc65969688)

[Test5: funkcia memory\_check 9](#_Toc65969689)

# Opis programu

## Program slúži na správu pamäte. Je naprogramovaný spôsobom explicitného zoznamu. Voľné bloky som spojené jednosmerne v spájaným zozname. To z dôvodu, aby hlavičky boli o 8 bytov pamäte menšie. Časová náročnosť programu je kvôli tomu síce o niečo väčšia, ale získali sme tím viac pamäte. Obzvlášť pri malých blokov je menšia hlavička výrazne viac oceniteľnejšia ako o niečo malo lepšia časová efektivita. Voľné bloky pri sebe sa spájajú a tým sa docieľuje menší počet väčších blokov.

## Hlavička bloku



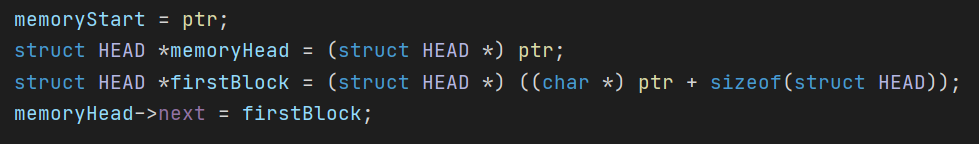
Hlavička bloku je štruktúra, ktorá má veľkosť 16 bytov. Je v nej uložený ukazovateľ na ďalší voľný blok a jeho veľkosť.

Na rozoznanie voľného bloku od obsadeného sa používa znamienko. Ak je veľkosť bloku záporná, ide o voľný blok. Ak je kladná, blok je už alokovaný.

# Opis funkcií

## memory\_init

Funkcia inicializuje pamäť. Je potrebná aby táto funkcia bola zavolaná pred ostatnými funkciami určenými na správu pamäte. Funkcia začiatku poľa vytvorí prvý blok, ktorý bude vždy ukazovať na voľný blok, ktorý sa bude používať pre alokovanie. Veľkosť tohto prvého bloku je podľa veľkosti pamäte, ktorá je zadaná pre program.



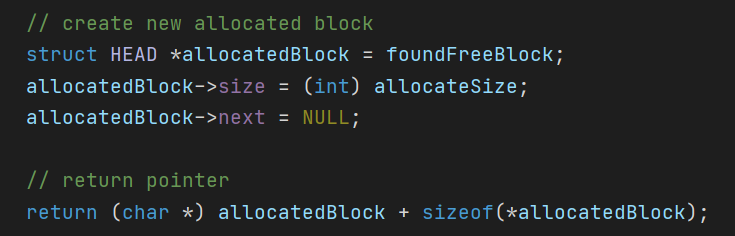
Obrázok 1 vytvorenie hlavičky, ktorá odkazuje na prvý voľný blok. Tento blok bude následne definovaný

## memory\_alloc

Funkcia začína vypočítanie potrebnej veľkosti. A to spočítaním vstupným parametrom, ktorý prezentuje požadovanú veľkosť, s veľkosťou hlavičky.

Túto potrebnú veľkosť použijeme ako argument pre funkciu **findBlock**.Ktorá nam za pomocí princípu **best fit** nájde v celom zozname najmenší, ale dostatočne veľký blok. Časová náročnosť je kvôli best fit vyššia(**O(n)**), ale vzhľadom na použitie explicitného zoznamu, ktorý uchováva iba voľné bloky to nemusí byť problém. Pretože voľných blokov nemusí byť veľa.

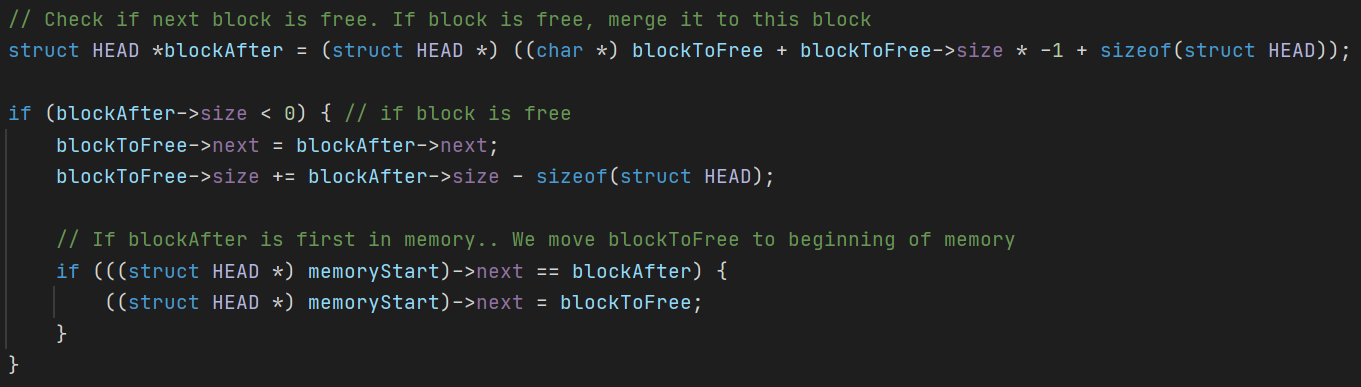
Ak požadovaná veľkosť je presne taká, aká je voľná veľkosť bloku, ktorý nám vrátila funkcia findBlock, tak sa tento blok priamo vyberie z listu. V opačnom prípade, ktorý bude určite častejší, sa voľný blok rozdelí. A to na požadovanú veľkosť a zbytok voľnej časti, ktorá sa môže použiť neskôr. Nový vytvorený alokovaný blok sa vráti ako ukazovateľ na začiatok payloadu.



Obrázok 2 vracanie novo vytvoreného alokovaného bloku

## memory\_free

Funkcia uvoľňuje už alokovanú časť pamäte. Vstupom pre túto funkciu je ukazovateľ na blok. Veľkosť bloku sa najprv nastaví na záporné číslo, čo symbolizuje to, že blok je znova voľný. Na konci funkcie sa pridá na začiatok zoznamu voľných bloková.

Pred tým sa ale overí čí blok pred ním alebo blok za ním je voľný. Ak áno, bloky sa spoja a tým vytvoria väčší blok pre budúcu alokáciu. Týmto výrazne zlepšíme pamäťovú efektívnosť, pretože vytvárame menej väčších blokov miesto veľa menších blokov. 

Obrázok 3 spájanie bloku v prípade, že nasledujúci blok je voľný

## memory\_check

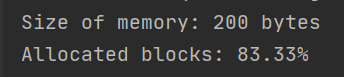
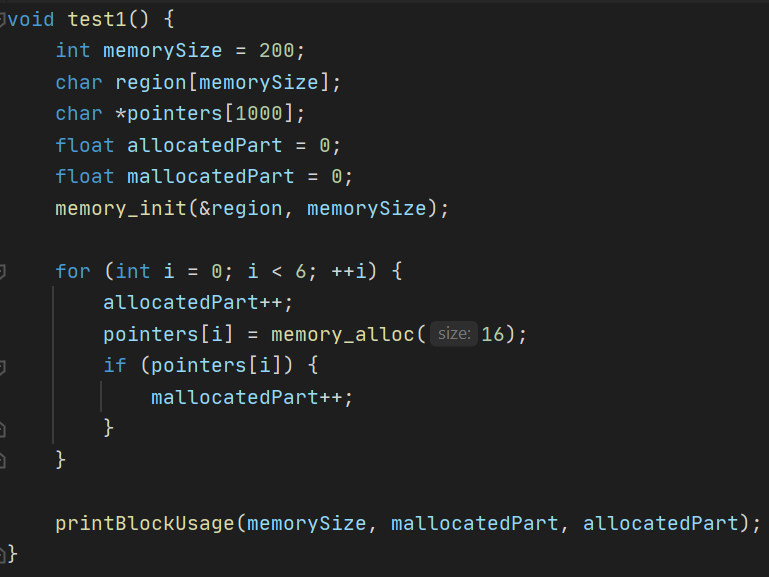
Funkcia, ktorá kontroluje, či sa ukazovateľ nachádza v alokovanom poli. Prechádza sa každým blokom od začiatku pamäte pokiaľ sa nenájde ukazovateľ, ktorý je ako vstupný parameter, vo voľnom poli. V opačnom prípade predpokladáme, že ukazovateľ je v alokovanom poli.



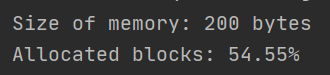
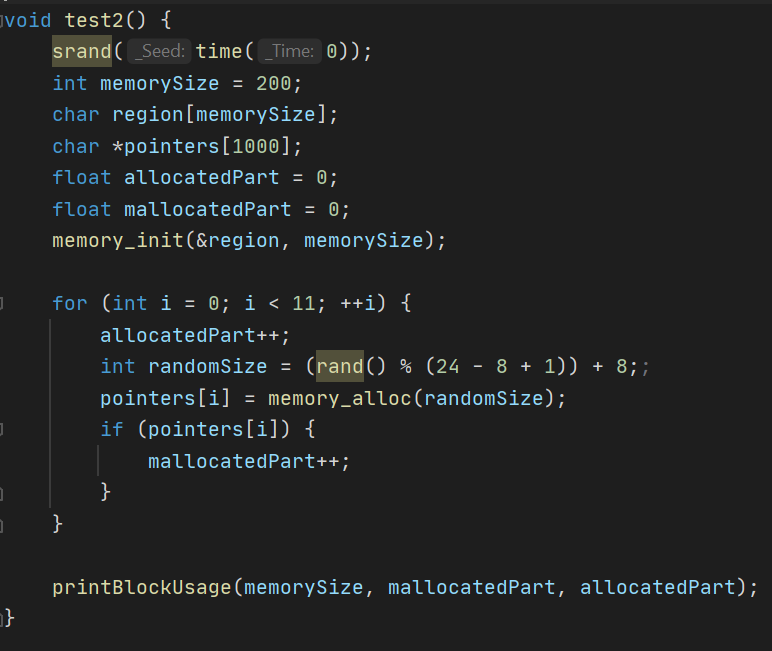
Obrázok 4 koniec while cyklu zabezpečujúci posúvanie sa na ďalší blok

# Testovanie programu

## Test1: Prideľovanie malých blokov rovnakej veľkosti

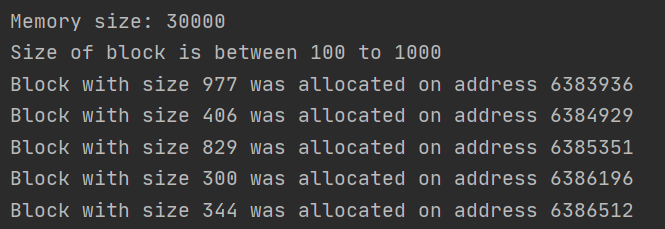
Funkcia testuje prideľovanie blokov rovnakej malej veľkosti(16) pre pamäť o veľkosti 200bitov.  
Veľkosť neje dostatočná pre šiesty blok a tak sa alokuje iba 5 zo 6tich blokov. To je 83.33%.

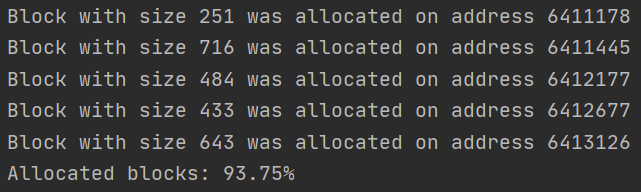
## Test2: prideľovanie blokov rôznej malej veľkosti

Funkcia testuje prideľovanie blokov náhodnej veľkosti medzi 8 až 24 bytov pre pamäť o veľkosti 200bitov. Výsledok je vždy rozdielna, vzhľadom na to, že sa nedá predpovedať aká veľkosť bytov bude. Ale môžeme predpokladať, že nikdy nebude 100%, keďže ak by mali všetky bloky minimálnu veľkosť, tak sa do pamäte aj tak nezmestia. 

## Test3: pridávanie rôzny počet náhodne veľkých blokov

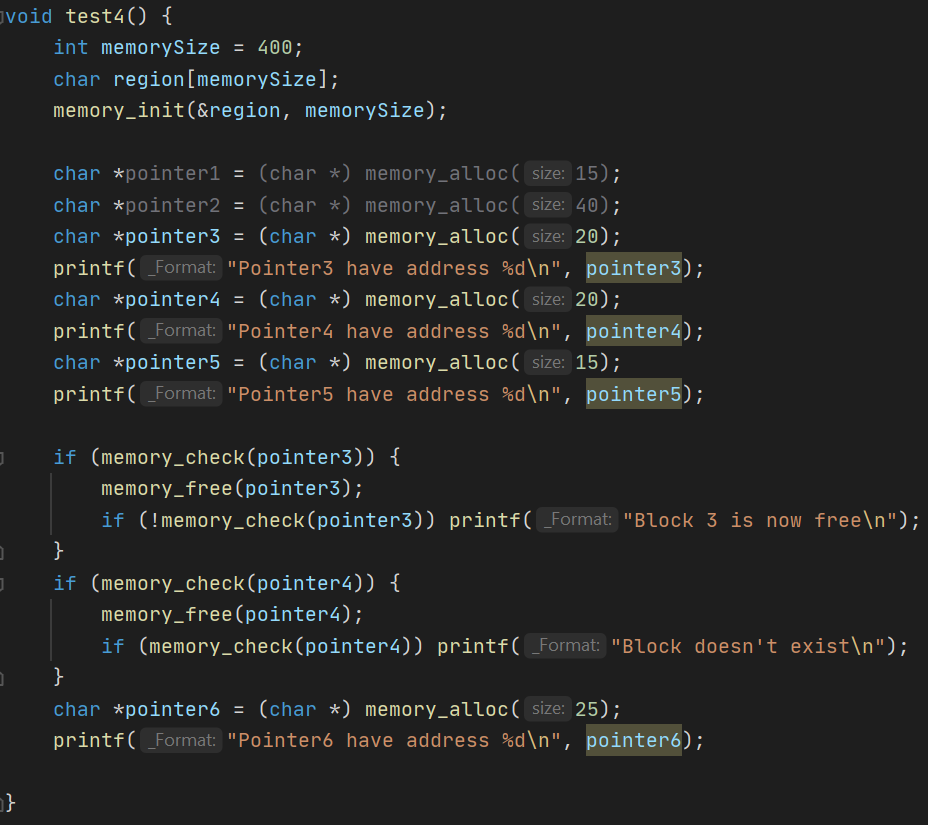
Pridávanie náhodne veľkých blokov do pamäte. Počet blokov je náhodný, pridáva sa kým je dostupná veľkosť. Obsadenosť blokov oproti ideálnemu počtu tu môže byť nižšia. To hlavne z dôvodu, že posledné pridané bloky môžu byť obrovské vzhľadom na celkovú veľkosť pamäte.



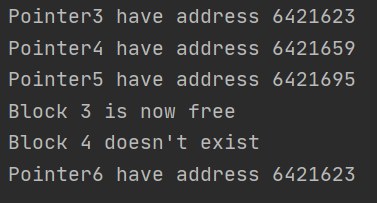


(Snímok so všetkými vypísanými blokmi som nezobrazil z dôvodu že ich je veľké množstvo)

## Test4: Uvoľňovanie a spájanie blokov

Test4 kontroluje konkrétny scenár. Testuje spájanie voľných blokov, ktoré sú vedľa seba. To z dôvodu, nech sa do pamäte zmestí viac väčších blokov. 

V tomto prípade alokujeme 5 ukazovateľov a potom dva z nich o veľkosti 20 uvoľníme. Funkciou memory\_check ešte overíme, že či ten blok, ktorý sa mal spojiť náhodou neexistuje.   
Spájanie blokov spôsobí to, že blok na ukazovateli 3 bude o veľkosti 40 a nový ukazovateľ 6 sa zmesti do daného bloku. Keby sa bloky nespájali, pamäť by mohla obsahovať veľa voľných malých blokov, čo by tiež spôsobovalo väčšiu fragmentáciu



## Test5: funkcia memory\_check

Test kontroluje scenáre s overovaním pamäte. Je otestované aj náhodne umiestnenie mimo hlavičku v poli. Či už v alokovanej oblasti alebo voľnej oblasti.