SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Zadanie 1

SPRÁVCA PAMÄTI

Anna Yuová

Predmet: Dátové štruktúry a algoritmy

Akademický rok: 2020/2021

Semester: letný

Správca pamäti

Pre vypracovanie môjho zadania som si vybrala metódu č.1 - spracovanie pomocou implicitného zoznamu. Na implementáciu som použila program DEV C++ (verzia 5.11) a kód je spustiteľný.

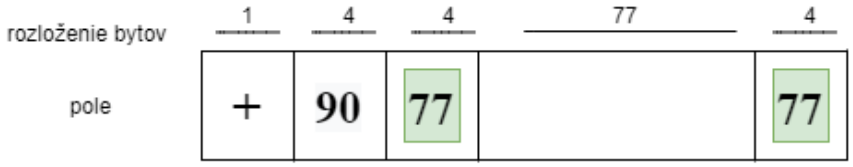
Opis môjho algoritmu

Na začiatku pri vytváraní pola som si vyhradila 1*char a 3*int čiže stratím 13B aj na hlavičku aj pätičku a aj si držím veľkosť pola aj to, kde moje pole začína. V programe mám jednu globálnu premennú (void *smernik) a je to smerník na začiatok prvej hlavičky celého poľa. Tento smerník si v programe pretypujem na dátový typ, ktorý potrebujem.

Memory_init()

-vytvorenie a pripravenie poľa so zadanou veľkosťou
-nastavenie do globálnej premennej (smerník) adresu na hlavičku – začiatok na tú časť poľa, s ktorou idem pracovať
-na prvom mieste je znak (char), aby som vedela, kde to pole začína
-za znakom je veľkosť poľa (int)
-za veľkosťou je hlavička(int) a pätička (int)
-v hlavičke a pätičke je na začiatku hodnota, ktorá sa dá alokovať

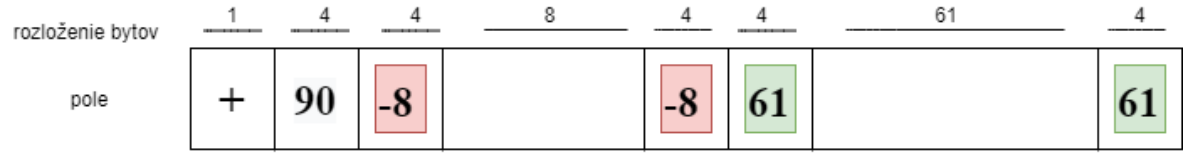
Napr. pre region[90]



Memory_alloc

-zavolanie funkcie najdiVolnyBlok() – prechádza celé pole – znak implicitného zoznamu
-voľné bloky hľadám metódou FirstFit – prvý vhodný voľný blok
-nastavím veľkosť a adresu hlavičky a pätičky
-alokovaná veľkosť hlavičky sa nastaví na zápornú hodnotu požadovanej veľkosti
-úspešná alokácia vráti smerník na úspešne alokovaný kus pamäte
-ak nie je dostatok miesta na alokáciu, funkcia vráti NULL

Napr. region[90], memory_alloc(8)



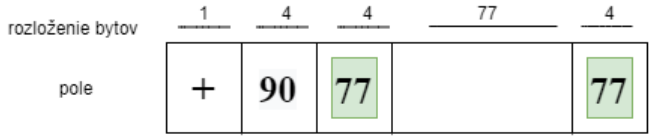
Memory_free

-nastavenie do premennej aktualnySmernik pointer na adresu, ktorú chcem uvoľniť (z argumentu funkcie)
-uloženie hodnoty na danej adrese, ktorú chcem uvoľniť
-kontrola či je tá hodnota < 0, čiže bola alokovaná a dá sa uvoľniť
-ak je splnená podmienka, zmení sa hodnota v hlavičke aj pätičke na absolútnu hodnotu
-potrebná kontrola či naokolo nie sú voľné bloky, ktoré by sa tiež dali spojiť od jedného veľkého voľného bloku
-kontrola pätičky pred hlavičkou a za pätičkou, ak sú kladné spojím ich

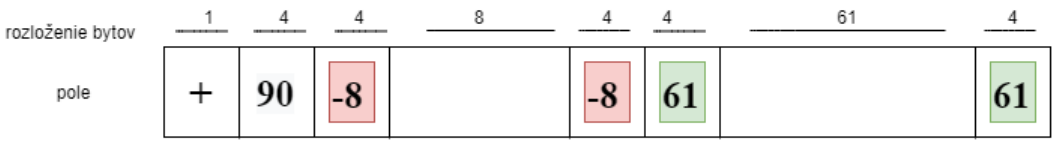
-vypočítam veľkosť spojenej časti, prepíšem hlavičku a pätičku a starú pätičku a hlavičku vynulujem

Napr. region[90]; int *ptr1 = memory_alloc(8); int *ptr2 = memory_alloc(6); memory_free(ptr1)

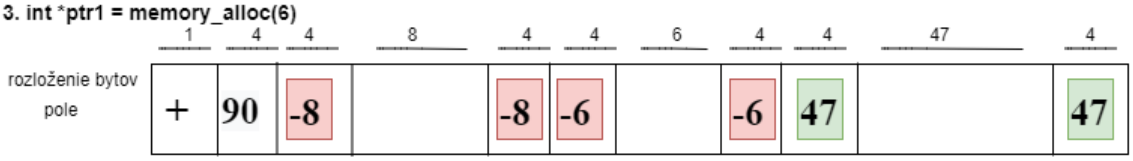
1. region[90]



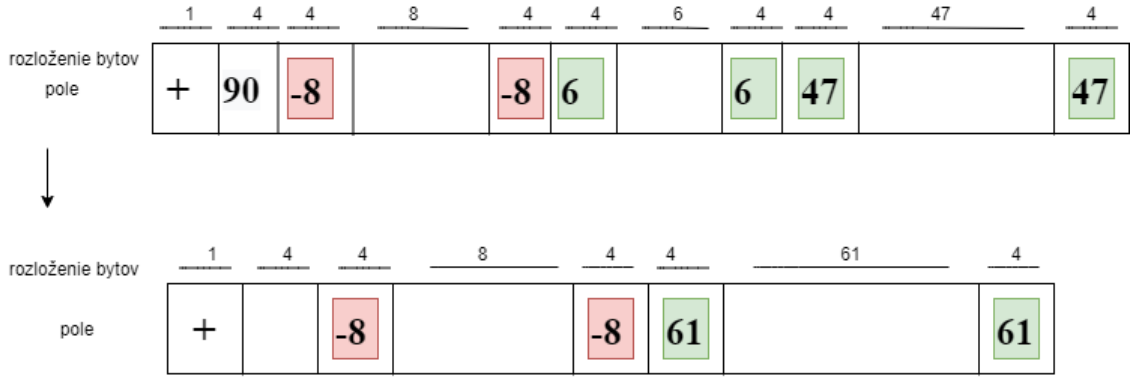
2. int *ptr1 = memory_alloc(8)



3. int *ptr1 = memory_alloc(6)

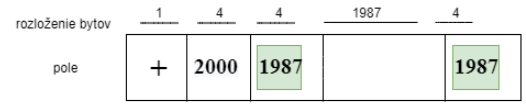


4. memory_free(ptr1)

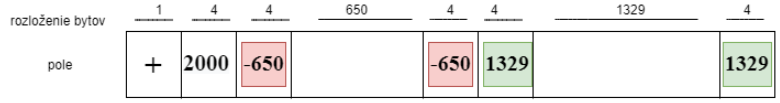


Napr. region[2000]; int *ptr1 = memory_alloc(650); int *ptr2 = memory_alloc(1000); memory_free(ptr2); int *ptr3 = memory_alloc(300); memory_free(ptr1); int *ptr4 = memory_alloc(250)

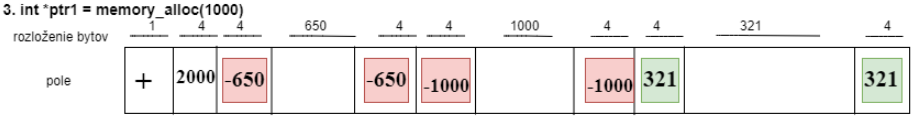
1. region[2000]



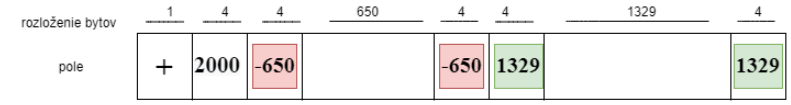
2. int *ptr1 = memory_alloc(650)



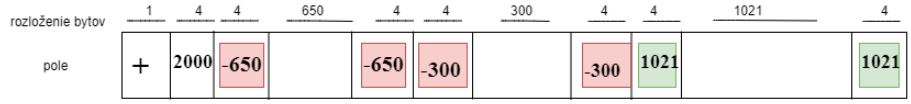
3. int *ptr1 = memory_alloc(1000)



4. memory_free(ptr2)

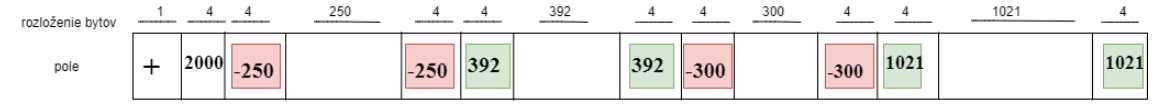


5. int *ptr3 = memory_alloc(300)



6. memory_free(ptr1)

7. int *ptr4 = memory_alloc(250)



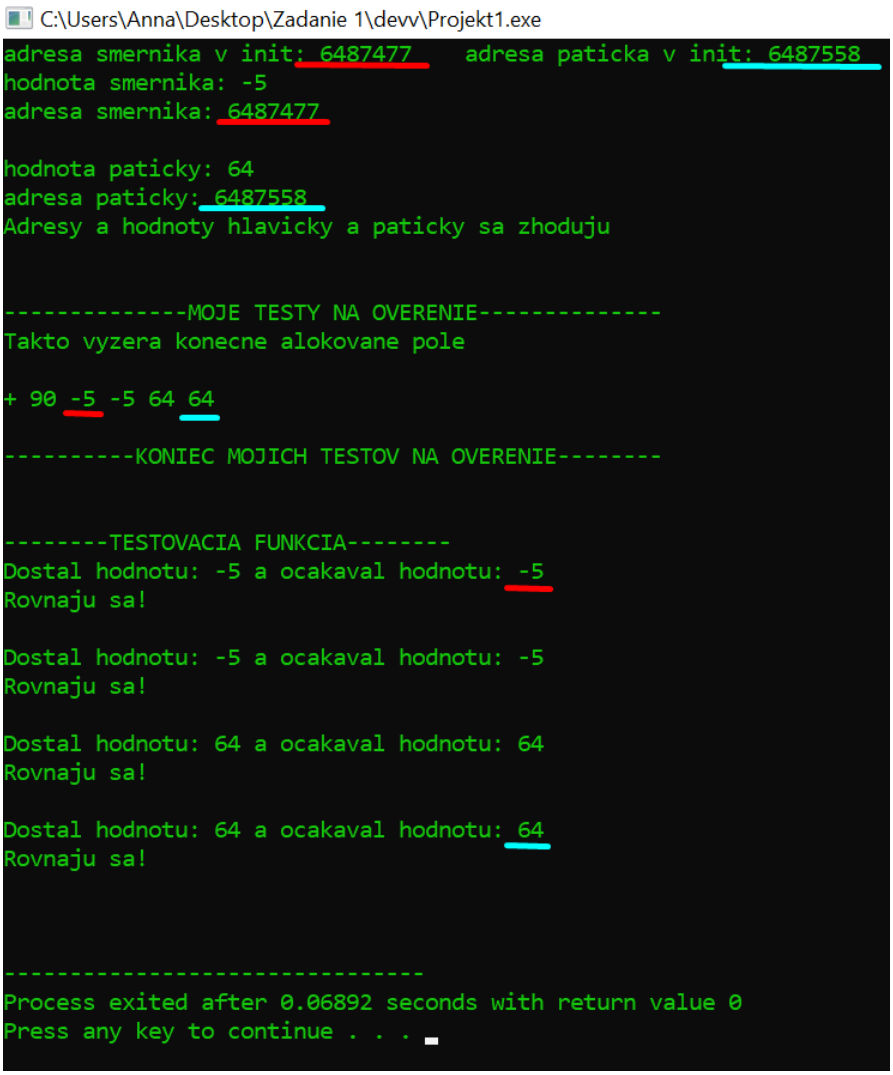
Vo funkcii **memory_check** skontrolujem či daný pointer v argumente je v rozsahu môjho poľa a či má zápornú hodnotu, teda či bol alokovaný. Ak to spĺňa tieto podmienky, funkcia vráti 1, inak 0.

Testovanie

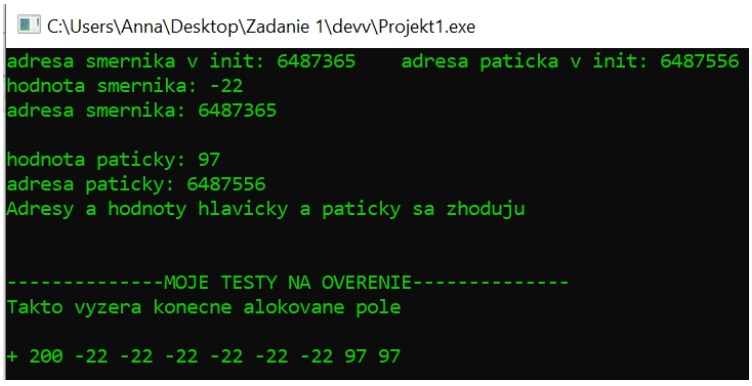
Vysvetlenie k výpisu:

Červená = ukáže adresu prvej hlavičky v poli (smerníka) v memory_init a tá sa zhoduje s adresou prvej hlavičky pri mojich testoch a aj v testovacej funkcii

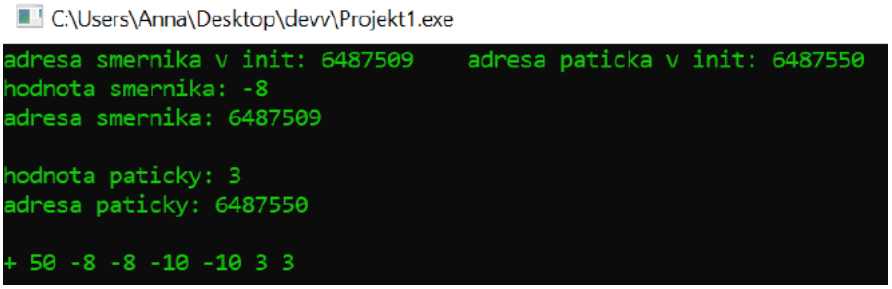
Modrá = ukáže adresu poslednej pätičky v memory_init a tá sa zhoduje s adresou poslednej pätičky pri mojich testoch a aj v testovacej funkcii



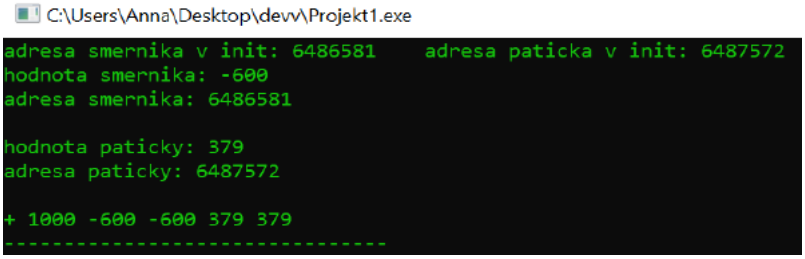
1. Testovala som rovnaké bloky malej veľkosti (8-24 bytov) pri použití celkových malých blokov pre správcu pamäte(50-200 bytov)



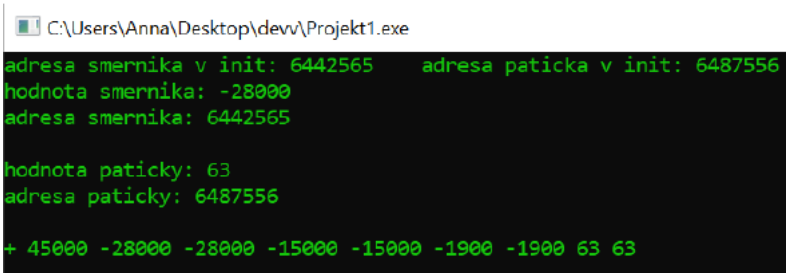
2. Testovala som nerovnaké bloky malej veľkosti (8-24 bytov) pri použití celkových malých blokov pre správcu pamäte(50-200 bytov)



3. Testovala som nerovnaké bloky väčšej veľkosti (500-5000bytov) pri použití väčších celkových blokov pre správcu pamäte (aspoň 1 000bytov)



4. Testovala som nerovnaké bloky väčšej aj menšej veľkosti (8- 50 000bytov) pri použití väčších celkových blokov pre správcu pamäte (aspoň 1000bytov)



Priestorová a časová zložitosť

Priestorová zložitosť

-Na začiatku pri vytváraní pola som si vyhradila 1*char a 3*int čiže stratím 13B aj na hlavičku aj pätičku a aj si držím veľkosť pola aj to, kde moje pole začína -> STRATA 13B
-Priestorová zložitosť je lineárna = O(n)

Časová zložitosť

-memory_init má časovú zložitosť O(1)
-memory_alloc má časovú zložitosť v najhoršom prípade O(n)
-memory_check má časovú zložisť O(1), iba overuje či je smerník vo vyhradenej pamäti a či ešte nebol uvoľnený, teda jeho hodnota je záporná
-memory_free má časovú zložitosť v najhoršom prípade O(n)