Adam Žúrek, ID: 79829

Dátové štruktúry a algoritmy

(Zadanie č. 1)

Obsah zadana: 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. Zadanie sa vypracúva v troch častiach:

1) Testovač prideľovania pamäti (do 3.10.2016 8:59)
2) Vlastný algoritmus prideľovania pamäti (do 10.10.2016 8:59)
3) Dokumentácia (do 17.10.2016 8:59)

- V prvej časti (odovzdanie do 3.10.2016 8:59) je potrebné implementovať testovač prideľovania pamäti. Teda, predpokladajte existenciu nejakého vytvoreného algoritmu na prideľovania pamäti (viď nižšie), vašou úlohou je vytvoriť program, ktorý túto implementáciu otestuje. Váš testovač by mali zahŕňať štandardné testy (inšpirujte sa v scenároch na konci tohto dokumentu), po prideľovaní bloku by váš testovač mal pamäť napĺňať rôznymi hodnotami a pri uvoľnení kontrolovať, či sú naplnené hodnoty stále správne naplnené (neboli prepísané iným blokom). Ako náhradu (kým nemáte vlastnú implementáciu z druhej časti zadania) môžete použiť napr. implementáciu využitím štandardných volaní knižnice stdlib.h pre správu pamäte: malloc a free.

- V druhej časti (odovzdanie do 10.10.2016 8:59) je vašou úlohou je implementovať nasledovné ŠTYRI funkcie využitím metódy implicitných zoznamov (alebo lepšej):

  ```
  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.

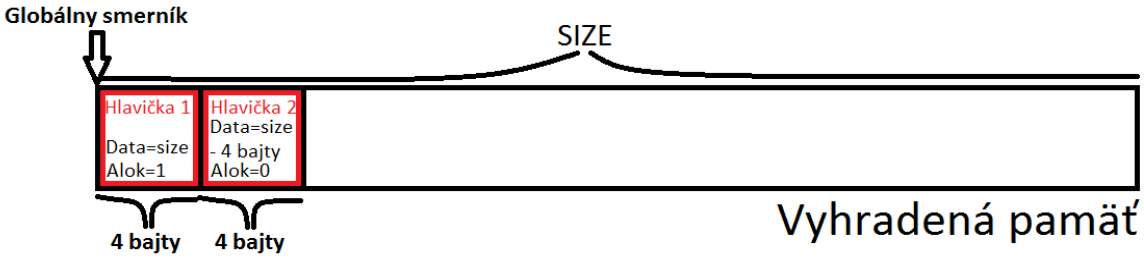
- V tretej časti (odovzdanie do 17.10.2016 8:59) odovzdávate textovú dokumentáciu, ktorá obsahuje hlavičku (kto, aké zadanie odovzdáva), stručný opis použitého algoritmu, s názornými nákresmi/obrázkami, a krátkymi ukážkami zdrojového kódu, vyberajte len kód na ktorý chcete extra upozorniť. Pri opise sa snažte dbať osobitý dôraz na zdôvodnenie správnosti vášho riešenia – teda, dôvody prečo je dobré/správne. Nakoniec musí technická dokumentácia obsahovať odhad výpočtovej (časovej) a priestorovej (pamäťovej) zložitosti vášho algoritmu. Celkovo musí byť cvičiacemu jasné, že viete čo ste spravili, že viete odôvodniť, že to je správne riešenie, a vedieť aké je to efektívne.

Vypracovanie: Funkcie som vypracoval využitím implicitných zoznamov a ako hlavičku jednotlivých alokovaných blokov som si zvolil typ premennej štruktúra.
Štruktúra má veľkosť 4 bajty, ktoré sú rozdelené do dvoch premenných typu unsigned (neznamienkový) int. Premenná obsahujúca dĺžku dát, ktoré patria konkrétnej hlavičke disponuje 31 bitmi. Druhá premenná obsahujúca informáciu o alokácii dát obsahuje jeden bit (0 pre uvoľnenú pamäť a 1 pre alokovanú).
Veľkosť pamäte, ktorú používateľ vyhradí funkciou memory_init si program musí tiež uchovať, preto sa pri jej zavolaní na začiatku vytvorí hlavička. Tá v sebe bude niesť informáciu o celej veľkosti pamäti.

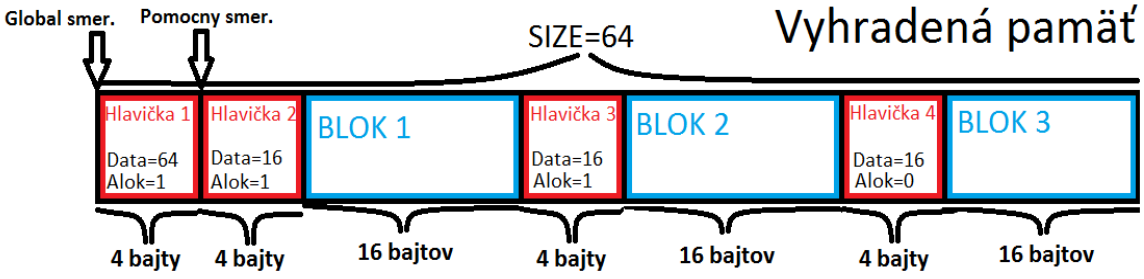
- Funkcia memory_init slúži na vyhradenie potrebnej pamäte, do ktorej bude program alokovať používateľom zadané bloky.
  Funkcia vytvorí dve hlavičky, kde prvá, ako som už spomenul v sebe uchováva veľkosť celej pamäte a druhá bude čakať na to, kým sa nespojí s alokovanou pamäťou.
  Nakoniec funkcia priradí globálnemu smerníku adresu ukazujúcu na prvú hlavičku vo vyhradenej pamäti.

  Časová zložitosť funkcie: O(1)

  Pamäťová zložitosť funkcie: P(1)



- Funkcia memory_alloc slúži na alokovanie blokov s rôznou veľkosťou do vyhradenej pamäte funkciou memory_init. Akonáhle sa funkcia zavolá, vytvorí sa pomocný smerník, ktorý „skáče“ z hlavičky na hlavičku, dokedy nepríde na koniec vyhradenej pamäte alebo nenájde nealokovanú hlavičku odkazujúcu na voľný blok vačší alebo



rovný ako potrebujeme alokovať. Presúvanie sa z hlavičky na hlavičku mu umožňuje uložená veľkosť pamäte v hlavičke, na ktorú každá odkazuje.
Pomocný smerník ukazuje na začiatku funkcie o veľkosť hlavičky (4 byty) ďalej za náš globálny smerník. Teraz sa pozrie, či je hlavička už alokovaná (v tomto prípade je). Teraz sa treba pozrieť na veľkosť dát, aké v hlavičke sú a o toľko skočí pomocný smerník na ďalšiu hlavičku, kde sa znovu pozrie, či je alokovaná (v tomto prípade je, tak skočí ešte). Po pozretí je hlavička voľná. Skontroluje či tam môže vložiť tak veľký blok, ako používateľ zadal. Dajme tomu, že zadal 16 bytov, blok sa tam akurát zmestí. V hlavičke sa prepíše, že miesto je už alokované a funkcia sa ukončí. Ďalší pokus o alokovanie by už skončil neúspechom. (Funkcia by vrátila NULL, pretože smerník dôjde až na úplný koniec vyhradenej pamäte, spýta sa globálneho smerníku, či už je na konci. Ak nič nenašiel a nemá kam ďalej ísť, pamäť je už celá zaplnená).

Ak by sme si zobrali situáciu z obrázka jedna, kde by sme začali alokovať od začiatku po zavolaní funkcie memory_init, vytvoril by sa pomocný smerník a hneď vidí, že hlavička nieje alokovaná a má dostatočne voľné miesto povedzme pre 50 bajtov. Pamäť sa alokuje a veľkosť sa priradí hlavičke a nastaví sa alokované na 1. Ale ak za alokovaným blokom ostala pamäť väčšia ako 4 bajty (veľkosť hlaičky) vytvorí sa tam hlavička, ktorá je nealokovaná a veľkosť v nej uložená ukazuje na celkovú zostávajúcu pamäť za ňou. Takto si funkcia memory_alloc predpripraví hlavičku na ďalšiu možnú alokáciu.

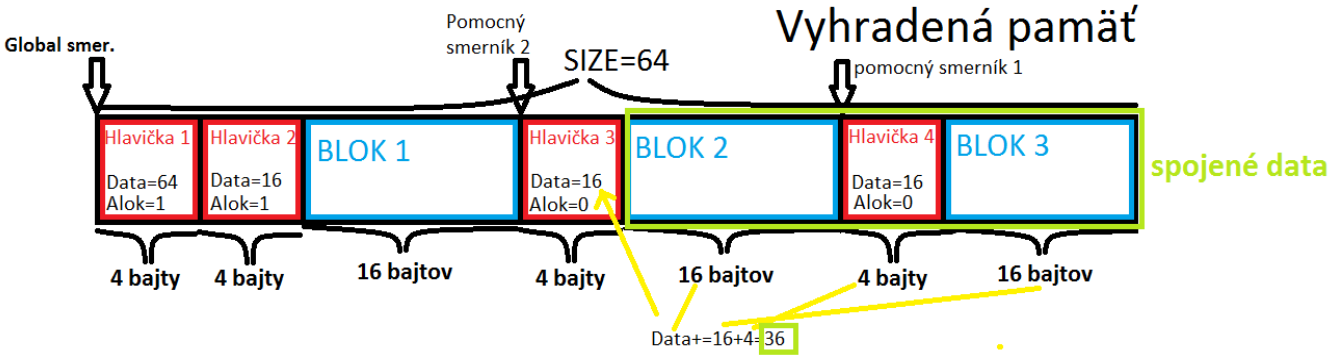
Časová zložitosť funkcie: O(n)

Pamäťová zložitosť funkcie: P(n)

- Funkcia memory_free po zavolaní vytvorí pomocný smerník, ktorému priradí adresu z parametra a ten blok hlavičky uvoľní. Následne po tejto akcii sa pomocný smerník posunie pred globálny smerník, ako vo funkcii memory_alloc a začne sa presúvať po hlavičkách a ak nájde volnú hlavičku (alokovaná=0) na tom mieste si funkcia vytvorí pomocný smerník 2 a posunie pomocný smerník 1 o hlavičku dopredu. Ak je uvoľnená aj tá, pomocný smerník 2 prepíše dáta v hlavičke, na ktorú ukazuje tak, že k nej pripočíta dáta na ktoré ukazuje hlavička so smerníkom pomocným 1 + veľkosť 4 bajty (veľkosť hlavičky) a tým ich spojí. Pomocný smerník 1 sa vráti na pozíciu pomocného smerníka 2 a opakuje postup ešte raz, kvôli možnosťi, že sme uvoľnili hlavičku medzi dvoma uvoľnenými hlavičkami. Vtedy by mohol spojiť až 2-krát. Samozrejme presúvanie smerníka kontrolujúceho dva po sebe uvoľnené bloky je ošetrený ako vo funkcii memory_alloc, aby sa nedostal za vyhradenú pamäť.

  Časová zložitosť funkcie: O(n)

  Pamäťová zložitosť funkcie: P(1)



Obrázok 3 ukazuje situáciu spájania dvoch za sebou uvoľnených blokov v momente sčítania dát.

- Funkcia memory_check slúži na skontrolovanie, či zadaný smerník ukazuje na alokovanú pamäť. Funkcia prechádza pomocným smerníkom všetky hlavičky a zakaždým kontroluje, či na tú istú adresu ukazuje aj hľadaný smerník. Ak áno a zároveň je hlavička alokovaná funkcia vráti 1. V opačnom prípade 0.

  Časová zložitosť funkcie: O(n)

  Pamäťová zložitosť funkcie: P(1)

Zhrnutie: Riešenie je v konečnom dôsledku správne, no dalo by sa ešte zefektívniť, ako napríklad meniť veľkosť hlavičky podľa pamäte, ktorú dostane, pretože je podľa môjho názoru pri alokácii rovnakých blokov po 8 bytoch je zvolenie 4-bajtovej hlavičky veľmi neefektívne.
Taktiež podľa môjho názoru existuje na defragmentáciu pamäte lepší spôsob, ako po každom uvoľnení prehľadávať celú pamäť.
Ako alternatívne postupy riešenia ma napadlo nahradiť štruktúry jedným smerníkom a zisťovanie, či je alokovaný by mohlo byť zaznamenané v parite alebo znamienku čísla.
Taktiež ma pre zefektívnenie defragmentácie napadol spôsob bez prehľadávania celej pamäte za pomoci pätičiek, no pamäť by tým bola veľmi neefektívne využitá.