Adam Miškove
ID: 103056

FIIT STU

Dátové štruktúry a algoritmy

Zadanie 1 - Správca pamäti

Adam Miškove

ID: 103056

2020/2021

Adam Miškove
ID: 103056

Zadanie 1 – Správca pamäti

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.
Konkrétnejšie je vašou úlohou je implementovať v programovacom jazyku C nasledovné ŠTYRI funkcie:

- 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.
Funkcia **memory_alloc** má poskytovať služby analogické štandardnému malloc. Teda, vstupné parametre sú
veľkosť požadovaného súvislého bloku pamäte a funkcia mu vráti: ukazovateľ na úspešne alokovaný kus
voľnej pamäte, ktorý sa vyhradil, alebo NULL, keď nie je možné súvislú pamäť požadovanej veľkosť vyhradiť.
Funkcia **memory_free** slúži na uvoľnenie vyhradeného bloku pamäti, podobne ako funkcia free. Funkcia
vráti 0, ak sa podarilo (funkcia zbehla úspešne) uvoľniť blok pamäti, inak vráti 1. Môžete predpokladať, že
parameter bude vždy platný ukazovateľ, vrátený z predchádzajúcich volaní funkcie **memory_alloc**, ktorý
ešte nebol uvoľnený.
Funkcia **memory_check** slúži na skontrolovanie, či parameter (ukazovateľ) je platný ukazovateľ, ktorý bol v
nejakom z predchádzajúcich volaní vrátený funkciou **memory_alloc** a zatiaľ nebol uvoľnený funkciou
memory_free. Funkcia vráti 0, ak je ukazovateľ neplatný, inak vráti 1.
Funkcia **memory_init** slúži na inicializáciu spravovanej voľnej pamäte. Predpokladajte, že funkcia sa volá
práve raz pred všetkými inými volaniami **memory_alloc**, **memory_free** a **memory_check**. Viď testovanie
nižšie. Ako vstupný parameter funkcie príde blok pamäte, ktorú môžete použiť pre organizovanie a aj
pridelenie voľnej pamäte. Vaše funkcie nemôžu používať globálne premenné okrem jednej globálnej
premennej na zapamätanie ukazovateľa na pamäť, ktorá vstupuje do funkcie **memory_init**. Ukazovatele,
ktoré prideľuje vaša funkcia **memory_alloc** musia byť výhradne z bloku pamäte, ktorá bola pridelená funkcii
memory_init.
Okrem implementácie samotných funkcií na správu pamäte je potrebné vaše riešenie dôkladne otestovať.
Vaše riešenie musí byť 100% korektné. Teda pokiaľ pridelíte pamäť funkciou **memory_alloc**, mala by byť
dostupná pre program (nemala by presahovať pôvodný blok, ani prekrývať doteraz pridelenú pamäť) a mali

by ste ju úspešne vedieť uvoľniť funkciou **memory_free**. Riešenie, ktoré nespĺňa tieto minimálne
požiadavky je hodnotené 0 bodmi. Testovanie implementujte vo funkcii **main** a výsledky testovania
dôkladne zdokumentuje. Zamerajte sa na nasledujúce scenáre:

- 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),
- 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),
- prideľovanie nerovnakých blokov väčšej veľkosti (veľkosti 500 až 5000 bytov) pri použití väčších
celkových blokov pre správcu pamäte (aspoň veľkosti 1000 bytov),
- 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).V testovacích scenároch okrem iného vyhodnoťte koľko % blokov sa vám podarilo alokovať oproti
ideálnemu riešeniu (bez vnútornej aj vonkajšej fragmentácie). Teda snažte sa prideľovať bloky až dovtedy,
kým v ideálnom riešení nebude veľkosť voľnej pamäte menšia ako najmenší možný blok podľa scenára.

Príklad jednoduchého testu:

```
#include <string.h>
int main()
{
char region[50];
//celkový blok pamäte o veľkosti 50 bytov
memory_init(region, 50);
char* pointer = (char*) memory_alloc(10);
//alokovaný blok o veľkosti 10 bytov
if (pointer)
memset(pointer, 0, 10);
if (pointer)
memory_free(pointer);
return 0;
}
```

Riešenie zadania sa odovzdáva do miesta odovzdania v AIS do stanoveného termínu (oneskorené
odovzdanie je prípustné len vo vážnych prípadoch, ako napr. choroba, o možnosti odovzdať zadanie
oneskorene rozhodne cvičiaci, príp. aj o bodovej penalizácii). Odovzdáva sa jeden **zip** archív, ktorý obsahuje
jeden zdrojový súbor s implementáciou a jeden súbor s dokumentáciou vo formáte **pdf**.
Pri implementácii zachovávajte určité konvencie písania prehľadných programov (pri odovzdávaní povedzte
cvičiacemu, aké konvencie ste pri písaní kódu dodržiavali) a zdrojový kód dôkladne okomentujte. Snažte sa,
aby to bolo na prvý pohľad pochopiteľné.

Adam Miškove
ID: 103056

Dokumentácia musí obsahovať 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, spôsob a vyhodnotenie testovania riešenia. 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
viete aké je to efektívne.

Hodnotenie

Môžete získať 15 bodov, **minimálna požiadavka 6 bodov**.
Jedno zaujímavé vylepšenie štandardného algoritmu je prispôsobiť dĺžku (počet bytov) hlavičky bloku podľa
jeho veľkosti. Každé funkčné vylepšenie cvičiaci zohľadní pri bodovaní.
Cvičiaci prideľuje body podľa kvality vypracovania. 8 bodov môžete získať za vlastný funkčný program
prideľovania pamäti (**aspoň základnú funkčnosť musí študent preukázať, inak 0 bodov**; metóda
implicitných zoznamov najviac 4 body, metóda explicitných zoznamov bez zoznamov blokov voľnej pamäti
podľa veľkosti najviac 6 bodov), 3 body za dokumentáciu (bez funkčnej implementácie 0 bodov), 4 body
môžete získať za testovanie (treba podrobne uviesť aj v dokumentácii). Body sú ovplyvnené aj prezentáciou
cvičiacemu (napr. keď neviete reagovať na otázky vzniká podozrenie, že to **nie je vaša práca, a teda je
hodnotená 0 bodov**).

Adam Miškove
ID: 103056

Doplnenie k zadaniu:

Dokumentácia musí obsahovať:

1. titulná strana
    a. názov inštitúcie,
    b. názov predmetu,
    c. názov zadania,
    d. meno a priezvisko,
    e. ais id,
    f. akademicky rok
    g. na každej strane v hlavičke: meno a priezvisko, ais id; v päte: názov zadania a číslovanie strán
2. znenie zadania
    a. to ktoré bolo vložené do AIS
    b. toto doplnenie
3. stručný opis algoritmu
    a. použite 2 spôsoby opisu
    b. doplňte ukážky zdrojového kódu na ktorý chcete extra upozorniť
4. testovanie
    a. scenár 1
        i. 8 do 50/100/200
        ii. 15 do 50/100/200
        iii. 24 do 50/100/200
    b. scenár 2
        i. rand (8-24) do 50/100/200 – zápis 5 hodnôt cyklicky do naplnenia pamäte
    c. scenár 3
        i. rand (500 – 5000) do 10 000 – zápis 5 hodnôt cyklicky do naplnenia pamäte
    d. scenár 4
        i. rand (8 – 50 000) do 100 000 – zápis 5 hodnôt cyklicky do naplnenia pamäte
    e. teoreticky výpočet alokácie vs. reálna hodnota
        i. pri výpočte neberte do úvahy vnútornú a vonkajšiu fragmentáciu
        ii. reálna je to čo alokujete
        iii. vyhodnoťte percentuálne
    f. nezabudnite na časovú zložitosť

Zdrojový kód:

1. memory_alloc
    a. blok spolu s réžiu dorovnať na najbližší vyšší násobok čísla 2
    b. blok/y na konci pamäte do ktorých nie je možné nič alokovať pripojiť k vedľajšej alokovanej časti
2. memory_free
    a. spájanie voľných blokov tak ako bolo uvedené na predná

Adam Miškove
ID: 103056

Stručný opis algoritmu:

Pre prácu s pamäťou používam pole typu unsigned char a zapisujem do neho informácie ohľadne stavu rôznych blokov pamäte a ich pozícií. Tieto informácie zapisujem zapisujem formou unsigned shortov (0-65535) do jednotlivých bytov, pričom tieto unsigned shorty rozdeľujem na jednotlivé byty, takže čísla od 0 do 255. Na toto som si vytvoril pomocnú funkciu putNumber();

```
void putNumber(unsigned char *ptr, unsigned short num){
    *(unsigned char*)(ptr) = num/256;
    *(unsigned char*)(ptr+1) = num%256;
}
```

Táto funkcia číslo potrebne rozdelí a zapíše na dva za sebou idúce byty.
Ak potrebujem z týchto dát znovu vytvoriť pôvodné číslo, použijem moju pomocnú funkciu getShortIntNumber();

```
unsigned short getShortIntNumber(unsigned char *ptr){
    return (unsigned short)((*ptr)*256)+(*(ptr+1));
}
```

Táto funkcia nám vráti pôvodné číslo, ktoré bolo na miesto zapísané.
Na manipuláciu s pamäťou používam explicitný spôsob, o ktorom sme si hovorili. "Pointery" (v tomto prípade indexy) sa nachádzajú iba v prázdnych blokoch pamäte.
Funkciu **memory_alloc** som si rozdelil na dve funkcie, pôvodnú funkciu, ktorá v podstate iba hľadá prvé miesto, kde sa alokovaná pamäť zmestí, a funkciu **allocate**, ktorá potrebné miesto alokuje a postará sa o všetky zmeny v informáciách v rámci mojej pamäte.
Funkciu **memory_free** som spravil tak, že nájde k nášmu uvoľnovanému bloku predošlý a nasledujúci voľný blok, teda bloky, ktorých informácie budú uvoľnením pamäte ovplyvnené. Potom už iba podľa rôznych prípadov, ktoré môžu nastať, prepíše informácie.
Voľné bloky spájam funkciou **fixFragments**, ktorá je volaná v **memory_free** po každom uvoľnení pamäte.
Funkcia **memory_check** nám zistí, či daný parameter(pointer) ukazuje na začiatok nami alokovaného bloku.
Funkcia **memory_init** pripraví hlavičky a pätičky pre danú časť pamäte.

Používam jednu primárnu hlavičku, pričom na prvých dvoch bytoch zapisujem veľkosť inicializovanej pamäte vo funkcii **memory_init**, a na druhých dvoch bytoch pointer(index) na nasledujúci voľný blok pamäte.
Pre alokovaný blok pamäte mám hlavičku a pätičku, oboje po dvoch bytoch, uchovávajúce veľkosť alokovanej pamäte pre daný blok.
Pre voľný blok mám navyše v hlavičke vyhradené dve dvojice bytov, pričom v prvej sa nachádza index pre predchádzajúci voľný blok pamäte, a v druhej pre nasledujúci.

Adam Miškove
ID: 103056

Testovanie:

Časová zložitosť:
memory_init -> O(n)
memory_alloc-> O(n)
memory_free-> O(n)
Memory_check-> O(n)

Scenár 1:

Všetky testy som skúšal a prešli mi.
Do väčšej hĺbky by som išiel na prezentácii tohto zadania na cvičení.

```
void testScenar1(int blockSize, int memorySize, unsigned char *str){
    unsigned char *ptr1 = (unsigned char*)200;          //len nieco aby tam nebol NULL
    float count=0;
    float efficiency;
    memory_init(str, memorySize);
    while(ptr1!=NULL){
        ptr1=memory_alloc(blockSize);
        if(ptr1!=NULL)count+=(float)blockSize;
    }
    for(int i=0;i<memorySize;i++){
        printf("|%5u",*(unsigned char*)(str+i));
    }
    printf("\n");
    for(int i=0;i<memorySize;i++) {
        printf("|%5u", i);
    }
    efficiency=((float)(count/(float)(memorySize))*100);
    printf("\n\nEfektivita tohto testu je: %.2f%%\n",efficiency);
}
```

Adam Miškove
ID: 103056

Scenár 2,3,4:
Takisto som skúšal všetky testy, a prešli mi. Samozrejme iba do veľkosti inicializovanej pamäte 65 535, pričom max mal byť 50 000.
Do väčšej hĺbky by som išiel na prezentácii tohto zadania na cvičení.

```
void testScenar234(int upper, int lower, int memorySize, unsigned char *str){
    unsigned char *ptr1 = (unsigned char*)200;
    float count=0;
    float efficiency;
    srand(time(0));
    int i=0, zvysok;
    int hodnota[5];    // vygenerujem 5 nahodnych hodnot z nasho intervalu
    for(int j=0;j<5;j++){
        hodnota[j]=randomNumber(lower,upper);
        printf("%d\n",hodnota[j]);
    }
    memory_init(str, memorySize);
    while(ptr1!=NULL){
        zvysok=i%5;
        ptr1=memory_alloc(hodnota[zvysok]);
        if(ptr1!=NULL)count+=(float)hodnota[zvysok];
        i++;
    }
    for(int i=0;i<memorySize;i++){
        printf("|%5u",*(unsigned char*)(str+i));
    }
    printf("\n");
    for(int i=0;i<memorySize;i++) {
        printf("|%5u", i);
    }
    efficiency=((float)(count/(float)(memorySize))*100);
    printf("\n\nEfektivita tohto testu je: %.2f%%\n",efficiency);
}
```

Výpočet pomeru alokovanej pamäte s celým inicializovaným blokom je implementovaný v testoch.