Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dátové štruktúry a algoritmy
Zadanie 1 – Správca pamäti

Martin Rudolf
AIS ID - 97029

Cvičenie: Pondelok 18.00
Cvičiaci: Ing. František Horvát, PhD.

2019/2020

Meno: Martin Rudolf
AIS - ID : 97029

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.
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**).

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. 8do 50/100/200
        ii. 15 do 50/100/200
        iii. 24do 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 10000–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 prednáške

Riešenie zadania:

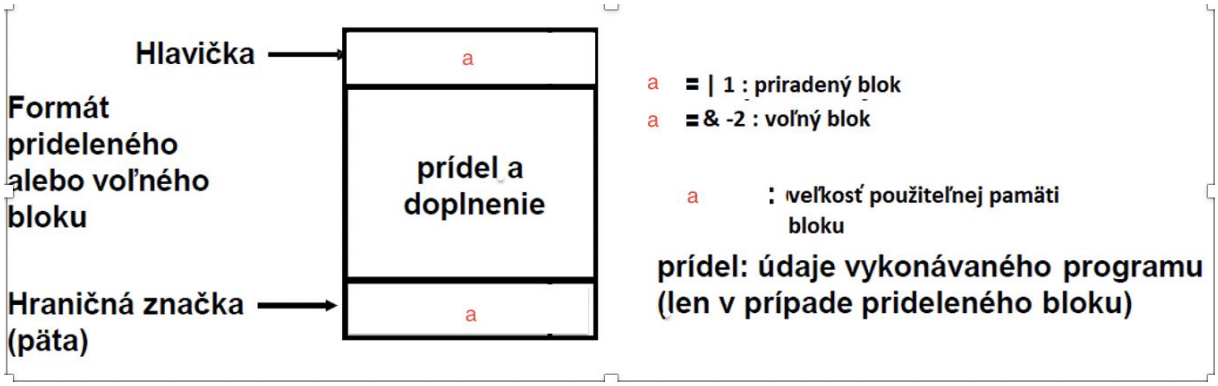
V tomto zadaní bolo treba implementovať už skôr spomenuté funkcie. Na implementovanie funkcie memory_alloc využívam implicitnú metódu vkladania s použitím dĺžok. Metódou first fit vkladám respektíve alokujem nový blok pamäte, veľkosť takéhoto bloku pamäte je vstupným argumentom funkcie, algoritmus prechádza celý vymedzený región, na prideľovanie pamäte, blok po bloku a ak nájde prvý voľný blok zodpovedajúci veľkosti požadovanej pamäti, obsadí ho a funkcia vráti pointer na alokovanú pamäť. Ak sa voľný blok s požadovanou veľkosťou v regióne nenachádza funkcia vráti NULL. Pri obsadzovaní voľného bloku, algoritmus zvýši požadovanú hodnotu o 1 a pomocou bitových operácií shift right a shift left zarovná hore na 2 následne binárna operácia or 1. Každý blok má svoju hlavičku a pätičku ktoré nesú informácie (veľkosť a obsadenosť) o danom bloku. Veľkosť hlavičky a pätičky sú 4 byty, takže ak chce užívateľ alokovať blok o veľkosti 10 bytov celkovo to zaberie 18 bytov z vyčleneného regiónu. Algoritmus funguje tak že ak po alokovaní posledného bloku pamäti ostane voľných menej ako 10 bytov pamäte tak budú pridelené k poslednému alokovanému bloku.

Vstupným argumentom funkcie memory_free je pointer na blok pamäte. Takýto pointer si funkcia najprv overí pomocou funkcie memory_check a ak je validný, bol vrátený funkciou memory_alloc a doposiaľ nebol uvoľnený, tak funkcia ho uvoľní. Algoritmus pomocou binárnej operácie and vykoná uvoľnenie: hodnota v hlavičke and -2, posunutie na pätičku, hodnota v pätičke and -2. Po tomto procese algoritmus kontroluje niekoľko scenárov. Skontroluje či predchádzajúci alebo nasledujúci blok je voľný, ak áno spojí ich, a tak vznikne nový blok s veľkosťou súčtu veľkostí susedných blokov, ich hlavičiek a pätičiek a aktuálne uvoľňovaného bloku. Ak sa podarilo uvoľniť požadovaný blok pamäte funkcia vráti 1.

Najjednoduchšou funkciou na implementáciu je funkcia memory_init. Táto funkcia dvoma riadkami kódu inicializuje požadovanú veľkosť pamäte na ktorú bude ukazovať pointer ktorý funkcia dostane ako vstupný argument, rovnako ako aj veľkosť.

Ako už bolo spomenuté funkcia memory_free využíva funkciu memory_check ktorá dostane na vstupe pointer a algoritmus prechádza bloky pamäte kým nenarazí na blok pamäte na ktorý ukazuje pointer zo vstupu, skontroluje či je blok obsadený ak áno vráti 1 ak je uvoľnený vráti 0.

Formát bloku:



Prechádzanie po blokoch:

```
while (pom != pamat_konec && (*pom & 1) || *pom < size) {
    (char*)pom += 2 * sizeof(unsigned int*) + (*pom & -2);
}
```

Testovanie

Na testovanie som si implementoval testovaciu funkciu test, ktorej vstupnými argumentami sú: pointer na región, pointer na pole pointerov, minimálna veľkosť regiónu, maximálna veľkosť regiónu, minimálna veľkosť bloku alokovaného bloku, maximálna veľkosť alokovaného bloku a hodnota cyklicky.

Na začiatku testovania prebehne inicializácia pomocou funkcie memory_init s náhodnou veľkosťou v rozsahu minimálna veľkosť regiónu a maximálna veľkosť regiónu. Ešte pred tým funkciou memset sa nastavia hodnoty vo vstupnom regione na 0.

Po inicializácií ak algoritmus vyhodnotí vstupnú hodnotu cyklicky, a ak sa rovná 1 vytvorí pole integerov a priradí do neho 5 rôznych náhodne vygenerovaný hodnôt, v rozsahu minimálna veľkosť bloku alokovaného bloku a maximálna veľkosť alokovaného bloku. Následne prebieha alokácia blokov s veľkosťou hodnou z poľa náhodne vygenerovaných. Algoritmus alokuje kým pamäť nie je ideálne plná. Vrátené pointre ukladám do poľa pointerov zo vstupu. Po alokácii prebieha uvoľňovanie pamäte. Úspešnosť alokovania je následne percentuálne vyjadrená ako podiel realne alokovaných blokov a ideálne alokovaných blokov vynásobený 100.

Vetva kde algoritmus vyhodnotí že cyklicky nesie hodnotu 0 robí to isté čo vetva testovania popísaná vyššie, no na alokovanie sa vždy vygeneruje nová náhodná hodnota v rozmedzí minimálna veľkosť bloku alokovaného bloku a maximálna veľkosť alokovaného bloku.

cyklicky = 1 :

```
while (allocated <= random_memory - min_random) {
    if (x > 4) {
        x = 0;
    }
    random = random_values[x++];
    if (allocated + random > random_memory)
        continue;
    allocated += random;
    allocated_count++;
    pointer[i] = (char*)memory_alloc(random);
    if (pointer[i]) {
        i++;
        mallocated_count++;
        mallocated += random;
    }
}
for (int j = 0; j < i; j++) {
    if (memory_check(pointer[j])) {
        memory_free(pointer[j]);
    }
    else {
        printf("Error: Zly memory check.\n");
    }
}
```

cyklicky = 0:

```
while (allocated <= random_memory - min) {
    random = (rand() % (max - min + 1)) + min;
    if (allocated + random > random_memory)
        continue;
    allocated += random;
    allocated_count++;
    pointer[i] = (char*)memory_alloc(random);
    if (pointer[i]) {
        i++;
        mallocated_count++;
        mallocated += random;
    }
}
for (int j = 0; j < i; j++) {
    if (memory_check(pointer[j])) {
        memory_free(pointer[j]);
    }
    else {
        printf("Error: Zly memory check.\n");
    }
}
```

Výsledky testovaných scenárov

Scenár 1 8 do 50/100/200, 15 do 50/100/200, 24 do 50/100/200

```
SCENAR 1
*-----------------------------------------------------------------------------------------------------------*
Velkost pamate o 50 bytov, velkost blokov 8: alokovanych 33.33% blokov (33.33% bytov).
Velkost pamate o 100 bytov, velkost blokov 8: alokovanych 50.00% blokov (50.00% bytov).
Velkost pamate o 200 bytov, velkost blokov 8: alokovanych 48.00% blokov (48.00% bytov).

Velkost pamate o 50 bytov, velkost blokov 15: alokovanych 33.33% blokov (33.33% bytov).
Velkost pamate o 100 bytov, velkost blokov 15: alokovanych 66.67% blokov (66.67% bytov).
Velkost pamate o 200 bytov, velkost blokov 15: alokovanych 61.54% blokov (61.54% bytov).

Velkost pamate o 50 bytov, velkost blokov 24: alokovanych 50.00% blokov (50.00% bytov).
Velkost pamate o 100 bytov, velkost blokov 24: alokovanych 75.00% blokov (75.00% bytov).
Velkost pamate o 200 bytov, velkost blokov 24: alokovanych 75.00% blokov (75.00% bytov).
```

Scenár 2 rand (8-24) do 50/100/200

```
SCENAR 2
*-----------------------------------------------------------------------------------------------------------*
Velkost pamate o 50 bytov, velkost blokov cyklicky 17 15 19 20 21: alokovanych 50.00% blokov (53.13% bytov).
Velkost pamate o 100 bytov, velkost blokov cyklicky 17 24 10 14 19: alokovanych 60.00% blokov (60.71% bytov).
Velkost pamate o 200 bytov, velkost blokov cyklicky 10 23 13 12 11: alokovanych 61.54% blokov (62.50% bytov).
```

Scenár 3 rand (500 –5000) do 10000

```
SCENAR 3
*-----------------------------------------------------------------------------------------------------------*
Velkost pamate o 10000 bytov, velkost blokov cyklicky 2916 3685 3246 2544 2315: alokovanych 100.00% blokov (100.00% bytov).
```

Scenár 4 rand(8 –50 000) do 100 000

```
SCENAR 4
*-----------------------------------------------------------------------------------------------------------*
Velkost pamate o 50000 bytov, velkost blokov cyklicky 7208 12571 31902 24750 28585: alokovanych 100.00% blokov (100.00% bytov).
```

Pozorovanie a záver

Implicitný zoznam s použitím dĺžok a metódou first fit, prehľadáva zoznam od začiatku a vyberie prvý voľný blok ktorý vyhovuje požadovanej veľkosti. Worst case Časovej zložitosti tejto metódy je lineárne úmerná celkovému počtu blokov. Uvoľňovanie blokov prebehne v konštantnom čase.