

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.

Zadanie sa vypracúva v dvoch častiach:

- 1) Vlastný algoritmus pridelovania pamäti (do 13.10.2019 21:00)
- 2) Dokumentácia (do 13.10.2019 21:00)

V prvej časti je vašou úlohou je implementovať nasledovné ŠTYRI funkcie využitím **metódy explicitných zoznamov alebo pokročilejšou**:

```
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 veľkosti 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 smerník z predchádzajúcich volaní vrátení funkciou **`memory_alloc`**, ktorý ešte nebol uvoľnený.

Funkcia **`memory_check`** slúži na skontrolovanie, či parameter (smerník) je platný smerník, ktorý bol v nejakom z predchádzajúcich volaní vrátení funkciou **`memory_alloc`** a zatiaľ nebol uvoľnený funkciou **`memory_free`**. Funkcia vráti 0, ak sa je smerník 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`**. Vid' 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 smerníku na pamäť, ktorá vstupuje do funkcie **`memory_init`**. Smerníky, ktoré prideluje vaša funkcia **`memory_alloc`** musia byť výhradne z bloku pamäte, ktorá bola pridelená funkcii **`memory_init`**.

Pred odovzdaním si vlastnú implementáciu dôkladne otestujte. Do testovača by ste mali odovzdávať už funkčnú verziu. Môžete síce do testovača odovzdať aj na viac krát, ale pokiaľ tam bude mať niekto 30 odovzdaných pokusov, niečo asi nie je v poriadku.

Ukážka testu:

```
#include <string.h>

int main()
{
    char region[50];
    memory_init(region, 50);
    char* pointer = (char*) memory_alloc(10);
    if (pointer)
        memset(pointer, 0, 10);
    if (pointer)
        memory_free(pointer);
    return 0;
}
```

V prvej časti odovzdávate len program. Program je napísaný v programovacom jazyku C, zachováva určité konvencie písania prehľadných programov (pri odovzdávaní povedzte cvičiacemu, aké konvencie ste pri písaní kódu dodržiavali). Snažte sa, aby to bolo na prvý pohľad pochopiteľné.

V druhej časti 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.

Čo keď nestihnem vypracovať a odovzdať všetky časti zadania do termínu?

Systém umožňuje vložiť vaše riešenie aj neskôr. V prípade omeškania pri odovzdaní zadania bezodkladne informujte svojho cvičiaceho, ktorý pri hodnotení omeškaného odovzdania môže vzhľadom na okolnosti odmietnuť hodnotiť zadanie alebo pristúpiť k bodovej penalizácii. V prípade, že zo zadania nezískate požadované minimálne 4 body, nesplnili ste požiadavky predmetu a končíte bez zápočtu.

Ako budeme vaše riešenie hodnotiť?

Môžete získať 10 bodov, minimálna požiadavka 4 body.

Implementácia metódou implicitných zoznamov nesplní minimálne požiadavky a je hodnotená 0 bodmi.

Implementácia metódou explicitných zoznamov bez zoznamov blokov voľnej pamäti podľa veľkosti je hodnotená najviac 8 bodmi.

Implementácia metódou explicitných zoznamov s využitím zoznamov blokov voľnej pamäti podľa veľkosti ale iná pokročilejšia implementácia je hodnotená najviac 10 bodmi.

Jedno zaujímavé vylepšenie štandardného algoritmu je prispôbiť dĺžku (počet bytov) hlavičky bloku podľa jeho veľkosti. Každé funkčné vylepšenie cvičiaci zohľadní pri bodovaní.

Cvičiaci prideluje body podľa kvality vypracovania. 5 bodov môžete získať za vlastný program pridelovania pamäti, 3 body za dokumentáciu, 2 body môžete získať za testovanie (treba 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). Body za dokumentáciu môžete dostať, len ak máte plne funkčnú implementáciu.

Program odovzdáte do testovača Turing, dostanete protokol o testovaní v rozličných scenároch.

- 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 prekryvať doteraz pridelenú pamäť), a mali by ste ju úspešne vedieť uvoľniť funkciou **memory_free**. Riešenie, ktoré nespĺňa tieto minimálne požiadavky je hodnotené 0 bodmi.
- Testovač Turing bude testovať riešenie vo viacerých scenároch, a bude sledovať:
 - korektnosť (tá je požadovaná vždy),
 - úspešnosť alokácií (koľko % z požadovaných alokácií prebehlo úspešne),
 - rýchlosť pridelovania (meranú ako rýchlosť behu vášho programu)
- Scenáre, ktoré môžeme testovať sú nasledovné:
 - použitie malých celkových blokov pre správcu pamäte (do 50 bytov, do 100 bytov, do 200 bytov) s nasledovnými dvoma scenármi. Inak sa budú používať väčšie celkové bloky (aspoň veľkosti 1000 bytov).
 - pridelovanie rovnakých blokov malej veľkosti (veľkosti 8 až 24 bytov)
 - pridelovanie nerovnakých blokov malej veľkosti (náhodné veľkosti 8 až 24 bytov)
 - pridelovanie nerovnakých blokov väčšej veľkosti (veľkosti 500 až 5000 bytov)
 - pridelovanie nerovnakých blokov malých a veľkých veľkostí (veľkosti od 8 bytov do 50 000)
 - Realizácia spájaného zoznamu (pridávanie, mazanie prvkov) – teda využitím vášho alokátora budeme riešiť úlohu (rôznych veľkostí) s využitím spájaného zoznamu. Požadujeme korektnosť, a tiež sa bude merať akú veľkú úlohu dokáže váš správca pamäte vyriešiť až kým nedokáže prideliť pamäť.

Budeme hodnotiť aj efektívnosť implementácie funkcií **memory_init**, **memory_alloc**, **memory_free** a **memory_check**. Efektívnejšia (časovo rýchlejšia alebo pamäťovo úspornejšia) implementácia dostane samozrejme viac bodov ako menej efektívna implementácia. **Pre získanie plného počtu bodov je potrebná implementácia metódou explicitných zoznamov voľných blokov alebo pokročilejšou!**

Dokumentácia musí obsahovať podrobný opis ako ste si svoje riešenie testovali. (vlastným testovačom na vlastných vstupoch, protokoly z testovača Turing sú len ako podklad k hodnoteniu, nie ako “váš” spôsob testovania).

Prajeme veľa úspechov.