პროგრამირების პარადიგმები - ფინალური - სესია 1 - 2022/23

ინსტრუქციები - აუცილებლად დაიცავით

ნაშრომის დატოვება

დესკტოპზე უნდა დაგხვდეთ დირექტორია რომლის სახელიც თქვენი უნივერსიტეტის ელ-ფოსტის პრეფიქსია. ამ დირექტორიაში უნდა დააკოპიროთ თქვენი ნაშრომები საგამოცდოს დატოვებამდე.
გირჩევთ თავიდანვე მოცემულ დირექტორიაში დააკოპიროთ საკითხები და მანდ იმუშაოთ ამოხსნებზე, რომ ბოლოს კოპირება არ დაგავიწყდეთ.

ტერმინალის გამოყენება

კოდის დასაწერად გირჩევთ გამოიყენოთ VSCode რედაქტორი რომელიც შეგიძლიათ იპოვოთ დესკტოპზე. ამოცანაზე სამუშაოდ მისი ფაილების გასახსნელად გამოიყენეთ: **File > Open Folder** და აირჩიეთ დესკტოპზე გადმოწერილი საგამოცდო საკითხების დირექტორია. კოდის დასაკომპილირებლად და ტესტების გასაშვებად შეგიძლიათ გამოიყენოთ VSCode-ში ჩაშენებული ტერმინალი: **Terminal > New Terminal**

ასევე შეგიძლიათ გამოიყენოთ Windows-ის სტანდარტული ტერმინალი. გახსენით საკითხის დირექტორია, sorted_kv_array მაგალითად, და ცარიელ ადგილას ჯერ დააჭირეთ **Control + Shift + Mouse-Right-Click** და შემდეგ აირჩიეთ **Open Command Prompt**

კოდის დაკომპილირებიდან მის გაშვებამდე კარგად დააკვირდით კომპილაცია წარმატებით დასრულდა თუ არა. კომპილაცია თუ ვერ შესრულდა, ძველი დაკომპილირებული პროგრამა (a.exe მაგალითად) უცვლელი ანუ ძველი რჩება. თუ გინდათ რომ დარწმუნდეთ ახალ დაკომპილირებულ პროგრამას ამოწმებთ გირჩევთ კომპილაციამდე ძველი კომპილაციის შედეგი წაშალოთ, მაგალითად ტერმინალში გაუშვით del a.exe

საკითხებთან ერთად მოცემული ტესტები არის მხოლოდ სამაგალითო, რათა გაგიადვილდეთ ნაშრომების შემოწმება. საბოლოო შეფასების დათვლისას ნაშრომები გასწორდება ტესტების სხვა სიმრავლეზე.

---

candy_machine - **ასემბლერი** (45 **ქულა** 200-**დან**)

თქვენი ამოცანაა candy_machine.c ფაილში არსებული C პროგრამირების ენაზე დაწერილი კოდი გადათარგმნოთ კურსზე გავლილ მანქანურ/ასემბლი კოდში. თარგმანი ჩაწერეთ candy_machine.txt ფაილში, სადაც უკვე გამოყოფილია ადგილი თითოეული C ენაზე დაწერილი ხაზის თარგმანისთვის.

თარგმნისას გაითვალისწინეთ რომ:

- არ მოგეთხოვებათ C კოდის ლოგიკური სისწორის შემოწმება ან მისი მნიშვნელობის გააზრება.
- არ მოგეთხოვებათ გენერირებული კოდის ოპტიმიზაცია.
- ყოველ ხაზზე რეგისტრების გამოყენება დაიწყეთ პირველიდან (R1), არ გამოიყენოთ წინა ხაზებში რეგისტრებში გადატანილი მნივნელობებ

SortedKVArray - **ჯენერიკები** (60 **ქულა** 200-**დან**)

თქვენი ამოცანაა იმპლემენტაცია გაუკეთოთ SortedKVArray მოცანემთა სტრუქტურას რომელსაც უნდა შეეძლოს (key, value) წყვილების შენახვა და უნდა ჰქონდეს შემდეგი ინტერფეისი:

- void SortedKVArrayInit(SortedKVArray* a, int key_size, int value_size, CompareFn key_cmp_fn, FreeFn key_free_fn, FreeFn value_free_fn); -- ინიციალიზაციას უნდა უკეთებდეს გადმოცემულ მასივს.
- void SortedKVArrayDestroy(SortedKVArray* a); -- უნდა ათავისიფულებდეს მასივის მიერ გამოყენებულ დინამიურ მეხსიერებას.
- void SortedKVArrayAdd(SortedKVArray* a, void *key, void *value); -- უნდა ამატებდეს მოცემულ (key, value) წყვილს მასივში. თუ მასივში მოცემული გასაღები უკვე გვხვდება მაშინ მასთან ასოცირებული მიმდინარე მნიშვნელობა უნდა წაიშალოს და მის ადგილას ახალი მნიშვნელობა ჩაიწეროს. **იღებს მფლობელობას ორივე key და value მისამართებზე**.
- bool SortedKVArrayRemove(SortedKVArray* a, void *key); -- მასივიდან უნდა აგდებდეს მოცემულ გასაღებს და მასთან ასოცირებულ მნიშვნელობას. თუ გასაღები მოიძებნა უნდა აბრუნებდეს true-ს, წინააღმდეგ შემთხვევაში კი false-ს. **არ იღებს მფლობელობას key მისამართებზე**.
- int SortedKVArraySize(SortedKVArray* a); -- უნდა აბრუნებდეს მასივის ზომას.
- void* SortedKVArrayGetKey(SortedKVArray* a, int index); -- უნდა დააბრუნოს მოცემულ ინდექსზე შენახული გასაღები. **არ გადასცემს მფლობელობას გამომძახებელს**.
- void* SortedKVArrayGetValue(SortedKVArray* a, void *key); -- უნდა აბრუნებდეს მოცემულ გასაღებთან ასოცირებული მნიშვნელობის მიმთითებელს. NULL-ს თუ ასეთი გასაღები ვერ მოიძებნა. **არ იღებს მფლობელობას key მისამართებზე. დაბრუნებული მისამართის მფლობელი რჩება გამოყენებულ SortedKVArray ობიექტს**.

ტესტირება

ფაილების კომპილაციისთვის გაუშვით *gcc .c*
ხოლო დაკომპილირებული ფაილის გასაშვებად: **./a.exe**

შეფასება

ტესტები შემდეგ კატეგორიებად იქნება დაყოფილი:

- 35% - მხოლოდ შეამოწმებს SortedKVArrayInit, SortedKVArrayDestroy, SortedKVArrayAdd და SortedKVArrayGet ფუნქციებს სადაც SortedKVArrayAdd მეთოდისთვის გადაცემული გასაღებები უნიკალურები იქნებიან.
- 35% - ზემოთ ხსენებულ მეთოდებთან ერთად შემოწმდება SortedKVArrayRemove, ანუ ყველა მეთოდი. ამ შემთხვევაშიც SortedKVArrayAdd მეთოდისთვის გადაცემული გასაღებები უნიკალურები იქნებიან.
- 30% - შემოწმდება ყველა მეთოდი. ამ შემთხვევაში SortedKVArrayAdd მეთოდისთვის გადაცემული გასაღებები უნიკალურები არ იქნებიან.

თითოეულ ტესტზე ინდივიდუალურად შემოწმდება თქვენი ამოხსნა სწორად იყენებს თუ არა მეხსიერებას. ყოველ წარმატებულად გავლილ ტესტზე რომლის შესრულების დროსაც თქვენი ნამუშევარი მეხსიერებას არასწორად გამოიყენებს დაგაკლდებათ ტესტისთვის განკუთვნილი ქულის 15%. სწორად გამოყენებაში იგულისხმება რომე აპლიკაცია უნდა იყენებდეს მხოლოდ მისთვის გამოყოფილ მეხსიერებას და აპლიკაციის დასრულებამდე ყველა დინამიურად გამოყოფილი მეხსერება უნდა გაათავისუფლოთ.

<stdlib.h>

- void* malloc(size_t size); -- Allocates size bytes of uninitialized storage.
- void *realloc(void *ptr, size_t new_size); -- Reallocates the given area of memory. It must be previously allocated by malloc(), calloc() or realloc() and not yet freed with a call to free or realloc. Otherwise, the results are undefined.
- void free(void* ptr); -- Deallocates the space previously allocated by malloc(), calloc(), aligned_alloc, (since C11) or realloc().

<string.h>

- void* memcpy(void *dest, const void *src, size_t count); -- Copies count characters from the object pointed to by src to the object pointed to by dest. Both objects are interpreted as arrays of unsigned char.
- void* memmove(void* dest, const void* src, size_t count); -- Copies count characters from the object pointed to by src to the object pointed to by dest. Both objects are interpreted as arrays of unsigned char. The objects may overlap: copying takes place as if the characters were copied to a temporary character array and then the characters were copied from the array to dest. The behavior is undefined if access occurs beyond the end of the dest array. The behavior is undefined if either dest or src is a null pointer.
- char * strdup(const char *str1); -- Returns a pointer to a null-terminated byte string, which is a duplicate of the string pointed to by str1. The returned pointer must be passed to free to avoid a memory leak.
- char *strndup(const char *str, size_t size); -- Returns a pointer to a null-terminated byte string, which contains copies of at most size bytes from the string pointed to by str. If the null terminator is not encountered in the first size bytes, it is added to the duplicated string.
- int strcmp (const char * str1, const char * str2); -- Compares the C string str1 to the C string str2. This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached. This function performs a binary comparison of the characters. For a function that takes into account locale-specific rules, see strcoll.
- int strncmp (const char * str1, const char * str2, size_t num); -- Compares up to num characters of the C string str1 to those of the C string str2. This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ, until a terminating null-character is reached, or until num characters match in both strings, whichever happens first.