Lista dostępnych zadań

#	Nazwa	Status	Trudność	Termin	Akcja
1	Kolokwium 02/02/2024 - zadanie na 3,0		☆☆☆☆☆ Punkty: 0	2024-02-02 zostało 2 godz	Wykonaj

#	Nazwa	Status	Trudność	Termin	Akcja

Przygotuj makro `CREATE_FUNCTION_COUNT_MAX_ONES_DISTANCE`, które tworzy funkcję zwracającą maksymalny dystans w badanej liczbie między bitami `1` rozdzielonymi bitami `0` lub stojącymi obok siebie w liczbie. Pozycję samych bitów `1` należy doliczyć do wyliczonego dystansu. Funkcja powinna być zbudowana w oparciu o następujący prototyp:

```
uint8_t count_max_ones_distance_##typ (typ a);
```

gdzie zamiast wyrazu `typ` powinien być wstawiony typ przekazany do makra. Na przykład, dla typu `char` funkcja powinna mieć następujący prototyp:

```
uint8_t count_max_ones_distance_char (char a);
```

Przykład zliczania:

```
11110011001 - powinno zwrócić 4
10111110001 - powinno zwrócić 5
10101010101 - powinno zwrócić 3
00000000011 - powinno zwrócić 2
00000000110 - powinno zwrócić 2
11100000011 - powinno zwrócić 8
00000000000 - powinno zwrócić 0
```

Zaimplementuj listę wiązaną jednokierunkową, mogącą przechowywać wyrazy wczytane z plików, wraz z informacją o ich liczbie wystąpień oraz łącznej liczbie ustawionych bitów w literach danego wyrazu. Konstrukcję listy oprzyj na następujących strukturach oraz API:

```
struct word_t
{
    char *word;
    uint8_t counter;
    uint8_t bit_counts;
};
```

gdzie:

- `word` - wyraz,
- `counter` - liczba wystąpień danego wyrazu w analizowanych plikach,
- `bit_counts` - maksymalny dystans w badanej liczbie między bitami `1` rozdzielonymi bitami `0` lub stojącymi obok siebie w liczbie (wykorzystaj funkcję `count_max_ones_distance_char`).

```
struct node_t{
    struct word_t *word;
    struct node_t *next;
};
```

gdzie:

- `next` - wskaźnik na następny węzeł listy, jeżeli nie ma następnego węzła to `NULL`,
- `word` - dane przechowywane w danym węzeł listy. Właścicielem bloku jest element `node_t`.

```
struct list_t{
    struct node_t *head;
};
```

gdzie:

- `head` - wskaźnik na pierwszy węzeł listy, jeżeli lista jest pusta to powinien być ustawiony na `NULL`,

Zaimplementuj następujące API:

```
int list_load_file(struct list_t **list, const char *filename);
void list_free(struct list_t *list);
void list_display(const struct list_t *list);
```

#	Nazwa	Status	Trudność	Termin	Akcja

Definicje wszystkich typów danych wraz z prototypami funkcji API należy dostarczyć w pliku `linked_list.h`, a deklarację w pliku `linked_list.c`.

```
int list_load_file(struct list_t **list, const char *filename);
```

Funkcje wczytuje wyrazy z pliku tekstowego `filename` i dodaje je do listy w kolejności w jakiej pojawiają się w pliku. W liście powinny znajdować się tylko unikalne wyrazy, jeżeli wczytany z pliku wyraz występuje już w liście to funkcja powinna zwiększyć licznik wystąpień tego wyrazu. Przyjmij, że wyrazem są dowolne ciąg znaków rozdzielone spacjami. Jeżeli do funkcji przekazana zostanie nieistniejąca lista (`*list == NULL`) , to funkcja powinna zaalokować pamięć i utworzyć nową listę, w przeciwnym przypadku nowo odczytane elementy powinny być dodawane do już istniejącej listy.

Funkcja może przydzielić tylko tyle pamięci, ile jest potrzebne do przechowania wszystkich danych.

Wartość zwracana:

- `0` - w przypadku sukcesu,
- `1` - w przypadku przekazania do funkcji błędnych parametrów,
- `2` - w przypadku, kiedy plik wejściowy nie istnieje,
- `3` – w przypadku niepowodzenia alokacji pamięci.

```
void list_free(struct list_t *list);
```

Funkcja zwalnia pamięć przydzieloną zarówno na listę, jak i wszystkie węzły oraz wyrazy przechowywane w tej liście. W przypadku przekazania błędnych danych funkcja nic nie robi.

```
void list_display(const struct list_t *list);
```

Funkcja wyświetla na konsoli zawartość listy, poszczególne węzłów powinny być wyświetlane w oddzielnych liniach, najpierw wyraz, następnie liczba wystąpień wyrazu oraz największ seria bitów ustawionych na `1` w tym wyrazie.

Napisz program, który w oparciu o przygotowane funkcje wczyta z plików podanych w parametrach programu wszystkie wyrazy, a następnie wyświetli je na konsoli. Jeżeli nie uda się otworzyć pliku program powinien wyświetlić komunikat: `Couldn't open file "nazwa pliku"`, i kontynuować wczytywanie pozostałych plików. Po wczytaniu wszystkich plików program powinien wyświetlić zawartość listy w taki sam sposób, jak funkcja `list_display`.

W przypadku, kiedy nie udało się przydzielić pamięci program powinien wyświetlić komunikat `Failed to allocate memory`, i zakończyć działanie z kodem błędu `3`, a w przypadku podania niewystarczającej liczby argumentów komunikat `Not enough arguments` i zakończyć działanie z kodem błędu `9`.

Przykład interakcji (brak argumentów):

```
$> ./main
Not enough arguments
```

Przykład interakcji (sukces): star_wars.txt bad.txt

Zawartość pliku star_wars.txt:

```
Blaster Clone Mandalorian Boba Fett Jabba Rancor Hologram Tauntaun Jawa Holocron Twi'lek Rodian Sarlacc Phantom Clone Wars The Force Dark Side Force Ghost TIE Fighter X-wing Star Destroyer Queen Amidala General Grievous BB-8 Snoke Womp Rat Sandcrawler Lightspeed Bounty Hunter
```

Zawartość pliku bad.txt:

```
Jedi Sith Lightsaber Millennium Falcon Skywalker Tatooine Coruscant Hoth Endor Yoda Rebel Empire Droid Wookiee R2-D2 C-3PO Hutt Naboo Mustafar Kashyyyk Jakku Alderaan Geonosis Bespin Dooku Kamino Leia Palpatine Obi-Wan Kenobi Padawan
```

#	Nazwa	Status	Trudność	Termin	Akcja

```
$> ./mainbad bad.txt race.txt skin star_wars.txt
Couldn't open file "race.txt"⏎
Couldn't open file "skin"⏎
Jedi 1 7⏎
Sith 1 9⏎
Lightsaber 1 24⏎
Millennium 1 23⏎
Falcon 1 15⏎
Skywalker 1 22⏎
Tatooine 1 21⏎
Coruscant 1 25⏎
Hoth 1 10⏎
Endor 1 13⏎
Yoda 1 10⏎
Rebel 1 9⏎
Empire 1 13⏎
Droid 1 12⏎
Wookiee 1 19⏎
R2-D2 1 8⏎
C-3PO 1 11⏎
Hutt 1 10⏎
Naboo 1 15⏎
Mustafar 1 20⏎
Kashyyyk 1 23⏎
Jakku 1 10⏎
Alderaan 1 17⏎
Geonosis 1 24⏎
Bespin 1 14⏎
Dooku 1 14⏎
Kamino 1 15⏎
Leia 1 8⏎
Palpatine 1 20⏎
Obi-Wan 1 18⏎
Kenobi 1 15⏎
Padawan 1 15⏎
Blaster 1 16⏎
Clone 2 13⏎
Mandalorian 1 27⏎
Boba 1 9⏎
Fett 1 10⏎
Jabba 1 9⏎
Rancor 1 15⏎
Hologram 1 21⏎
Tauntaun 1 20⏎
Jawa 1 8⏎
Holocron 1 23⏎
Twi'lek 1 15⏎
Rodian 1 14⏎
Sarlacc 1 15⏎
Phantom 1 17⏎
Wars 1 11⏎
The 1 5⏎
Force 2 13⏎
Dark 1 8⏎
Side 1 8⏎
Ghost 1 15⏎
TIE 1 3⏎
Fighter 1 17⏎
X-wing 1 15⏎
Star 1 10⏎
Destroyer 1 25⏎
Queen 1 11⏎
Amidala 1 13⏎
General 1 17⏎
Grievous 1 23⏎
BB-8 1 7⏎
Snoke 1 13⏎
Womp 1 12⏎
Rat 1 6⏎
```

#	Nazwa	Status	Trudność	Termin	Akcja

```
Sandcrawler 1 26⏎
Lightspeed 1 24⏎
Bounty 1 18⏎
Hunter 1 15⏎
```

Uwagi

- Wszystkie operacje na tablicach powinny być wykonywane za pomocą wskaźników, a nie operatora `[]`!.
- Wszystkie operacje na bitach powinny odbywać się za pomocą operatorów bitowych, a nie `%` i `*` !.
- W programie można zadeklarować pomocniczą tablice do wczytywania danych z pliku, deklaracja tablicy **musi** wyglądać następująco: `char text[21];`

2	Kolokwium 02/02/2024 - rozszerzenie na 4,0		☆☆☆☆☆ Punkty: 0	2024-02-02 zostało 2 godz	Wykonaj

Rozszerz poprzednie zadanie o następujące funkcje (**ocena 4,0**):

```
void list_sort(struct list_t *list, int (*comp)(const struct node_t *, const struct node_t *));
int node_comp_string(const struct node_t *a, const struct node_t *b);
```

Funkcja `list_sort` sortuje elementy listy rosnąco zgodnie z wartościami zwracanymi przez funkcję `comp`. Funkcja przekazana jako argument `comp` powinna zwracać następujące wartości:

- mniejsze od 0 - gdy wartość argumentu pierwszego jest mniejsza od argumentu drugiego;
- równe 0 - gdy wartość argumentu pierwszego jest równa wartości argumentu drugiego;
- większe od 0 - gdy wartość argumentu pierwszego jest większa od argumentu drugiego.

W przypadku przekazania do funkcji błędnych argumentów funkcja nie powinna nic robić. W funkcji nie można alokować dodatkowej pamięci, ani zmieniać zawartości węzłów, sortowanie może się tylko odbywać poprzez zamianę wartości wskaźnika `next`.

Do przetestowania działania swojej funkcji przygotuj funkcję `node_comp_string` porównującą wyrazy znajdujące się w analizowanych węzłach, funkcja powinna zwrócić:

- 0, jeżeli wyrazy w obu węzłach są takie same,
- mniejszą od 0, jeśli pierwszy niezgodny znak w węźle1 jest niższy (w ASCII) niż w węźle2.
- większą od 0, jeśli pierwszy niezgodny znak w węźle1 jest większy (w ASCII) niż w węźle2.

Wszystkie deklaracje funkcji oraz definicje struktur powinny znajdować się w pliku `linked_list.h`, a definicje funkcji w pliku `linked_list.c`.

Uwagi

- Wszystkie operacje na tablicach powinny być wykonywane za pomocą wskaźników, a nie operatora `[]`!.
- W programie można zadeklarować pomocniczą tablice do wczytywania danych z pliku, deklaracja tablicy **musi** wyglądać następująco: `char text[21];`

3	Kolokwium 02/02/2024 - rozszerzenie na 5,0		☆☆☆☆☆ Punkty: 0	2024-02-02 zostało 2 godz	Wykonaj

#	Nazwa	Status	Trudność	Termin	Akcja

Rozszerz poprzednie zadanie o następujące funkcje (**ocena 5,0**):

Zaimplementuj następujące funkcje w API listy jednokierunkowej. Odpowiadają one za sortowanie wielu elementów do listy jednokierunkowej.

```
int list_puch_back_multiple(struct list_t *list, int N, ...);
```

Funkcja dodaje do listy list *N* wyrazów przekazanych w parametrach funkcji. Wyrazy powinny być dodawane w takiej samej kolejności, w jakiej występują w parametrach. Jeżeli jakiś wyraz już był w liście, to powinien zostać zwiększony dla niego licznik wystąpień. Funkcja zwraca:

- -1 – w przypadku przekazania do funkcji błędnych argumentów,
- liczbę dodanych wyrazów do listy (wliczając również wyrazy, dla których był tylko zwiększony licznik wystąpień).

Wszystkie deklaracje funkcji oraz definicje struktur powinny znajdować się w pliku `linked_list.h`, a definicje funkcji w pliku `linked_list.c`.

Uwagi

- Wszystkie operacje na tablicach powinny być wykonywane za pomocą wskaźników, a nie operatora `[]`!.
- W programie można zadeklarować pomocniczą tablice do wczytywania danych z pliku, deklaracja tablicy **musi** wyglądać następująco: `char text[21];`