ProgramowanieProceduralne

IS_L7

W ramach zajęć przygotowałam dwa zadania. Ponieważ zadanie pierwsze może być pewnym wyzwaniem, dlatego zasady są następujące:
suma punktów z całych zajęć to 13, natomiast suma punktów z zadań to 16.

1.(8) Proszę napisać program pozwalający na obsługę listy jednokierunkowej dowolnego typu i przetestować dla listy przechowującej wartości typu **`double`** oraz łańcuchy znaków.
Dane do testowania:

```
double numbers[] = { 7.4, 1.3, 14.5, 0.1, -1.0, 2.3, 1,2, 43.0, 2.0, -4.7, 5.8 };
char *strings[] = { "Zorro", "Alex", "Celine", "Bill", "Forest", "Dexter"};
```

```
typedef void (*free_fun)(void *);//definicja funkcji zwalniającej pamięć usuwanego elementu - wykorzystane, dla złożonych typów
```

```
typedef int (*list_list)(void *);//definicja funkcji wypisującej pamięć zaalokowaną w pojedynczym elemencie listy
typedef struct _listNode {
  void *data; //wskaźnik do alokowanej pamięci, pozwalającej na przechowywanie  wartości dowolnego typu
  struct _listNode *next; //wskaźnik do kolejnego elementu
} listNode; // struktura używana przez funkcje implementujące

typedef struct {//struktura opisująca listę
  int list_len;//ilość elementów w liście
  int el_size; //rozmiar elementu listy
  listNode *head; //wskaźnik do początku listy
  listNode *tail; //wskaźnik do końca listy
  free_fun freeFn; //funkcja zwalniająca pamięć usuwanego elementu
} list;
```

Proszę dopisać brakujące funkcje obsługujące listę

- **`void list_new(list *list, int elementSize, free_fun free_free);//inicjalizacja listy`**
  funkcja przyjmuje trzy argumenty: wskaźnik do listy **`*list`**, rozmiar przechowywanych/alokowanych elementów **`elementSize`** oraz wskaźnik do funkcji **`free_free`**, która ma zostać wywołana dla każdego elementu, który ma zostać usunięty - wartości przechowywane/alokowane w elementach mogą być złożone i wymagają stosownej funkcji do zwolnienia przydzielonej pamięci; dla typów odkładanych na stosie wskaźnik może być **`NULL`**

```
void list_new(list *list, int elementSize, free_fun free_free)
{
  assert(elementSize > 0);
  list->list_len = 0;
  list->el_size = elementSize;
  list->head = list->tail = NULL;
  list->freeFn = free_free;
}
```

- **`void list_free(list *list); //zwolnienie pamięci listy`**
- **`void list_front(list *list, void *element); //dodanie elementu na początek listy`**

```
void list_front(list *list, void *element)//argumenty to: wskaźnik do listy oraz wskaźnik do wartości, którą ma przechowywać nowotworzony element listy
{
  listNode *node = malloc(sizeof(listNode)); //tworzenie elementu listy
  node->data = malloc(list->el_size); //alokacja pamięci na wartość, która ma przechowywać element listy
  memcpy(node->data, element, list->el_size);// skopiowanie zawartości do zaalokowanej pamięci

  node->next = list->head;
  list->head = node;

  if(!list->tail) {
    list->tail = list->head;
  }

  list->list_len++;
```

```
}
```

- **`void list_end(list *list, void *element); //dodanie elementu na koniec listy`**
- **`void list_all(list *list, list_list iter_list); //wypisanie elementów listy`**
  funkcja przyjmuje dwa argumenty: wskaźnik do listy **`*list`**, oraz wskaźnik do funkcji **`iter_list`**, która wywoływana jest dla każdego elementu listy **`listNode`**i pozwala wypisać (w zdefiniowany tam sposób) zawartość "przypiętą" do wskaźnika **`void *data`** . Funkcja **`iter_list`** zwraca 0 jeżeli wystąpił błąd w wypisywaniu zawartości elementu
- **`void list_first(list *list, void *element); //przechwycenie, wypisanie i usuniecie elementu z początku listy`**
- **`void list_last(list *list, void *element); //przechwycenie, wypisanie i usuniecie elementu z końca listy`**

Program działa dla listy typu `double` w zakresie zaimplementowanych funkcji - tworzenie listy, wypisywanie, zwalnianie pamięci. Dla listy `stringów` trzeba uzupełnić wypisywanie i zwalnianie pamięci. W ramach zadania należy dopisać wszystkie brakujące funkcje, oraz przetestować je dla obu list

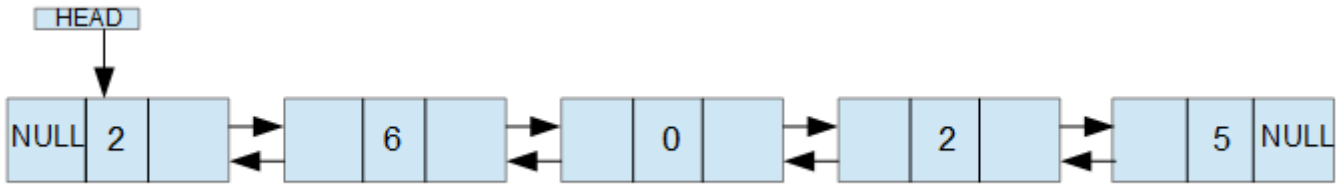
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

typedef void (*free_fun)(void *);
typedef int (*list_list)(void *);//definicja funkcji wypisującej pamięć zaalokowaną w pojedynczym elemencie listy

typedef struct _listNode {
  void *data; //wskaźnik do alokowanej pamięci, pozwalającej na przechowywanie  wartości dowolnego typu
  struct _listNode *next; //wskaźnik do kolejnego elementu
} listNode; // struktura używana przez funkcje implementujące


typedef struct {//struktura opisująca listę
  int list_len;//ilość elementów w liście
  int el_size; //rozmiar elementu listy
  listNode *head; //wskaźnik do początku listy
  listNode *tail; //wskaźnik do końca listy
  free_fun freeFn; //funkcja zwalniająca pamięć usuwanego elementu
} list;

void list_new(list *list, int elementSize, free_fun free_free)
{
  assert(elementSize > 0);
  list->list_len = 0;
  list->el_size = elementSize;
  list->head = list->tail = NULL;
  list->freeFn = free_free;
}

void list_front(list *list, void *element)//argumenty to: wskaźnik do listy oraz wskaźnik do wartości, którą ma
przechowywać nowotworzony element listy
{
  listNode *node = malloc(sizeof(listNode)); //tworzenie elementu listy
  node->data = malloc(list->el_size); //alokacja pamięci na wartość, która ma przechowywać element listy
  memcpy(node->data, element, list->el_size);// skopiowanie zawartości do zaalokowanej pamięci

  node->next = list->head;
  list->head = node;

  if(!list->tail) {
    list->tail = list->head;
  }

  list->list_len++;
}

int iterate_double(void *data);//wypisywanie double "doczepianych" do elementów listy
int iterate_string(void *data);//wypisywanie stringów "doczepianych" do elementów listy
void free_string(void *data);//zwalnianie pamieci na stringi "doczepiane" do elementów listy

void list_all(list *list, list_list iter_list)
{
  assert(iter_list != NULL);

  listNode *node = list->head;
  int result = 1;
  while(node != NULL && result) {
    result = iter_list(node->data);
    node = node->next;
  }
}

void list_free(list *list)
{
  listNode *current;
  while(list->head != NULL) {
    current = list->head;
    list->head = current->next;

    if(list->freeFn) { //np. dla alokowanych stringów
      list->freeFn(current->data);
    }

    free(current->data);//dla pamięci, nie alokowanej
```

```
    free(current);
  }
}

void list_double()
{
  double numbers[] = { 7.4, 1.3, 14.5, 0.1, -1.0, 2.3, 1,2, 43.0, 2.0, -4.7, 5.8 };

  int i;
  list list_d;
  list_new(&list_d, sizeof(double), NULL);

  for(i = 0; i <= 10; i++) {
    list_front(&list_d, numbers+i);
  }

  list_all(&list_d, iterate_double);
 //testowanie pozostałych funkcji
  list_free(&list_d);
  printf("Successfully freed  numbers...\n");
}

void list_strings()
{
  int numNames = 5;
  const char *names[] = { "Zorro", "Alex", "Celine", "Bill", "Forest", "Dexter"};

  int i;
  list list_s;
  list_new(&list_s, sizeof(char *), free_string);

  char *name;
  for(i = 0; i < numNames; i++) {
    name = strdup(names[i]);//Funkcja strdup() zwraca wskaźnik do nowego łańcucha, który stanowi kopię łańcucha s. Pamięć
dla nowego łańcucha jest przydzielana za pomocą malloc() i może być zwolniona za pomocą free().
    list_front(&list_s, &name);
  }

  list_all(&list_s, iterate_string);
 //testowanie pozostałych funkcji
  list_free(&list_s);
  printf("Successfully freed %d strings...\n", numNames);
}

 int iterate_double(void *data)
{
  printf("Found value: %f\n", *(double *)data);
  return 1;
}

 int iterate_string(void *data)
{
  //nalezy uzupełnić
  return 1;
}

void free_string(void *data)
{
  //nalezy uzupełnić
}
int main(int argc, char *argv[])
{

  list_double();
  list_strings();
}
```

2. (8) Dana jest struktura

```
   struct tnode {
      int value;
      struct tnode *next;
      struct tnode *prev;
    };
```

Proszę zaimplementować obsługę llisty dwukierunkowej:



- wypisywanie listy - **`void print_list(struct tnode*);`**
- dodawanie elementu do listy na początku - **`struct tnode* add_first(struct tnode* head, struct tnode* el);`**
- dodawanie elementu do listy na końcu - **`struct tnode* add_last(struct tnode* head, struct tnode* el);`**
- wyciąganie elementu z listy (wczytanie wartości klucza do wyboru elementu), jeżeli mamy kilka elementów o podanym kluczu zwracamy pierwszy -

  `struct tnode* del_el(struct tnode** head, int var);`
- dokładanie w porządku rosnącym - **`void add_sort(struct tnode** head1, struct tnode* el);`**
- podział listy na dwie listy - wartości parzyste i nieparzyste z wykorzystaniem wcześniej zaimplementowaych funkcji - **`struct tnode* div_list(struct tnode** head);`**
- implementacja rekurencyjnego algorytmu szybkiego sortowania na liscie dwukierunkowej z wykorzystaniem wcześniej zaimplementowaych funkcji- **`void sort(struct tnode** head);`**
- zwalnianie listy - **`void free_list(struct tnode**);`**

Status przesłanego zadania

Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Nieocenione
Termin oddania	poniedziałek, 8 czerwca 2020, 14:25
Pozostały czas	Zadanie zostało złożone 8 min. 18 sek. przed terminem
Ostatnio modyfikowane	poniedziałek, 8 czerwca 2020, 14:16
Przesyłane pliki	2.c — 8 czerwca 2020, 14:16
Komentarz do przesłanego zadania	Komentarze (0)

◄ LAB_15 Przejdź do... zad_1 przydział pamięci ►