Wstęp do Informatyki i Programowania
Ćwiczenia: Lista 5

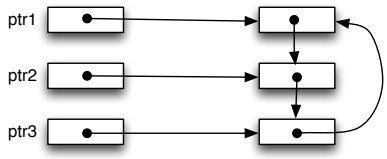
Przemysław Kobylański

Zadanie 1

Rozpatrzmy następujący fragment programu w C:

```
struct Node {struct Node *ptr;};
struct Node *ptr1 = malloc(sizeof(struct Node));
struct Node *ptr2 = malloc(sizeof(struct Node));
struct Node *ptr3 = malloc(sizeof(struct Node));
ptr1->ptr = ptr2;
ptr2->ptr = ptr3;
ptr3->ptr = ptr1;
```

Po jego wykonaniu wskaźniki przyjmują wartości takie jak przedstawiono na poniższym rysunku:



Pozmieniaj na rysunku strzałki tak aby przedstawiały wartości wskaźników po wykonaniu kolejnych instrukcji:

```
ptr1->ptr = ptr1->ptr->ptr;
ptr2->ptr = ptr2->ptr->ptr;
ptr3->ptr = ptr3->ptr->ptr;
```

Zadanie 2

W zadaniach od 2 do 4 przyjmij, że w programie zadeklarowane są następujące dwa typy:

```
struct Node
{
  int value;
```

```
  struct Node *next;
};

typedef struct Node *List;
```

Załóżmy, że lista liczb całkowitych, której elementy reprezentowane są strukturą `Node`, ma elementy uporządkowane niemalejąco, tzn. jeśli `ptr` wskazuje dowolny element ale nie ostatni, tzn. `ptr->next` ma wartość różną od `NULL`, to

`ptr->value <= ptr->next->value`.

Napisz funkcję `void dopisz_niemalejaco(List *ptr, int i)`, która wstawia na listę, dla której `*ptr` jest wskazaniem na pierwszy jej element, wartość całkowitą `i` z zachowaniem niemalejącego porządku elementów listy.

Zadanie 3

Załóżmy, że elementy listy reprezentowane są strukturami `Node` (tak jak zdefiniowano w zadaniu 2), przy czym elementy nie muszą być uporządkowane niemalejąco, tzn. kolejność wartości na liście jest dowolna.

Napisz funkcję `void usun(List *ptr, int i)`, która usuwa z listy, dla której `*ptr` jest wskazaniem na pierwszy jej element, wszystkie wystąpienia wartości całkowitej `i`.

Uwagi

1. Czy nie zapomniałeś/aś zwolnić pamięci zajmowanej przez niepotrzebną już strukturę za pomocą funkcji `free`.
2. Czy Twoja funkcja dobrze radzi sobie w przypadku, gdy usuwana wartość występuje na liście więcej niż jeden raz?
3. Czy Twoja funkcja dobrze sobie radzi w przypadku, gdy po usunięciu wszystkich wartości `i` lista stanie się pusta?

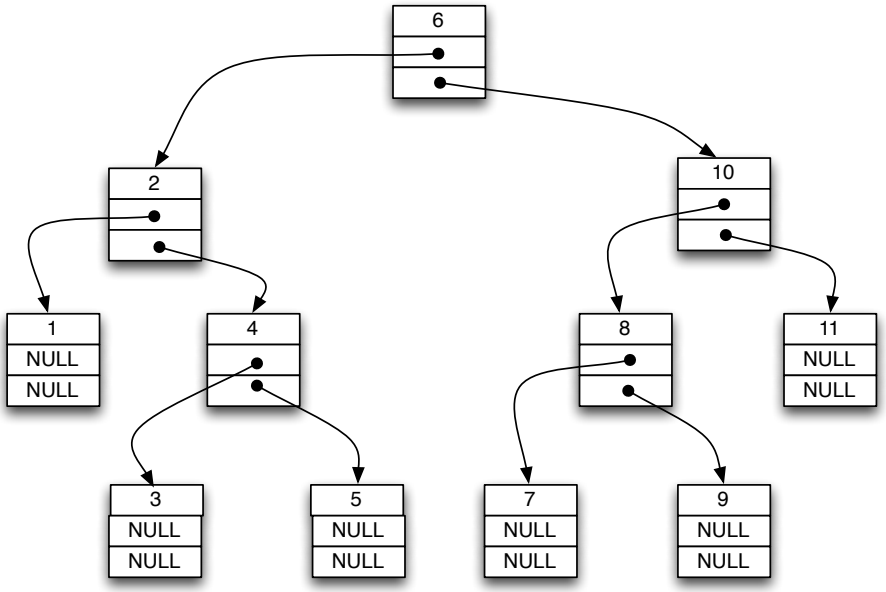
Zadanie 4*

Załóżmy, że elementy listy reprezentowane są strukturami `Node` (tak jak zdefiniowano w zadaniu 2).

Napisz funkcję `odwroc_liste(List *ptr)`, która odwraca kolejność elementów na liście wskazywanej przez `*ptr`.

Zadanie 5

Rozpatrzmy strukturę drzewa, w którego każdym węźle przechowywana jest liczba całkowita oraz dwa wskaźniki na lewe (`left`) i prawe (`right`) poddrzewo:



W powyższym drzewie węzeł zawierający wartość 6 nazywa się korzeniem drzewa. Każdy inny węzeł drzewa można osiągnąć idąc od korzenia i przechodząc do lewego lub prawego poddrzewa.

Dla przykładu, do węzła zawierającego wartość 7 dojdziemy idąc kolejno do prawego, lewego i lewego poddrzewa.

Zdefiniuj typy `struct Node` (dla przechowywania węzła drzewa) i `Tree` (będący wskazaniem na strukturę `Node`).

W zadaniach od 6 do 11 będziesz korzystał(a) z tych dwóch typów.

Zadanie 6

Napisz funkcję `int liczba_wezlow(Tree ptr)`, której wartością jest liczba węzłów w drzewie wskazywanym przez `ptr`.

Zadanie 7

Napisz funkcję `int liczba_lisci(Tree ptr)`, której wartością jest liczba liści w drzewie wskazywanym przez `ptr`.

Zadanie 8

Napisz funkcję `int wysokosc(Tree ptr)`, której wartością jest wysokość drzewa wskazywanego przez `ptr`.

Zadanie 9

Napisz funkcję `void usun_drzewo(Tree *ptr)`, która całkowicie usuwa z pamięci drzewo o korzeniu wskazywanym przez `*ptr`.

Uwaga

Czy Twoja funkcja poprawnie umieszcza stałą `NULL` w miejscu wskazywanym przez parametr `ptr`?

Zadanie 10*

Spróbuj rozwiązać poprzednie zadanie bez użycia rekurencji (jedynie iteracyjnie). Jaka jest czasowa złożoność obliczeniowa Twojej funkcji usuwającej drzewo o n węzłach?

Zadanie 11**

Spróbuj rozwiązać zadanie 9 bez użycia rekurencji ale niech Twoja funkcja ma czasową złożoność obliczeniową rzędu $O(n)$.

Postaraj się nie używać za dużo dodatkowej pamięci. Jeśli drzewo ma n węzłów, to rozmiar dodatkowych danych potrzebnych do usunięcia drzewa powinien być rzędu $O(1)$, tzn. stała.