Zadanie: LPRO
Przetwarzanie list

Wprawki WP 2022, Runda 8: listy wskaźnikowe. *6.01.2023*

Dla omawianego na zajęciach typu list prostych

```
struct lista {
  int val;
  struct lista *next;
};
typedef struct lista Tlista;
```

napisz procedurę

```
przetworz(char[] program, Tlista **lista_ptr)
```

która przetworzy listę prostą `*lista_ptr` zgodnie z instrukcjami w programie. Program to ciąg dwuznakowych instrukcji (zakończony znakiem `\0`, jak każdy standardowy łańcuch znaków w C), każda instrukcja składa się ze znaku komendy (litera) i znaku argumentu (cyfra). Komendy wykonywane są względem bieżącego elementu listy, zwanego dalej kursorem, na początku jest to startowy element listy.

Instrukcje są następujące:

- **Ik** wstaw nowy węzeł o zawartości k przed kursorem, kursor pozostaje bez zmian (czyli po wstawionym węźle);
- **Dk** usuń k elementów począwszy od kursora, ustaw kursor po wszystkich usuniętych elementach (D0 nic nie robi);
- **Fk** przesuń kursor o k elementów naprzód (F0 nic nie robi);
- **Rk** odwróć kolejność k elementów począwszy od elementu wskazywanego przez kursor, ustaw kursor po elemencie początkowo wskazywanym przez kursor (R0 nic nie robi, R1 to to samo co F1);
- **Mk** przestaw element wskazywany przez kursor k elementów dalej i ustaw kursor po nim (M0 to to samo co F1, M1 to to samo co R2);
- **Hk** ustaw kursor na k-ty element od początku listy (licząc od zera).

Uwaga! Możesz założyć, że lista ma wystarczającą liczbę elementów do zrealizowania wszystkich instrukcji w programie. W trakcie wykonywania programu kursor może stać za ostatnim elementem listy (czyli wskazywać na NULL), możliwe są wtedy tylko instrukcje Ik, D0, F0, R0, Hk.

Na przykład lista

```
10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> *
```

po wykonaniu kolejnych instrukcji programu

```
"D2F1R5I1I9H1M5D1H0I0"
```

będzie zmieniać się następująco (nawias kwadratowy oznacza pozycję kursora):

```
[10] -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> *
D2
[12] -> 13 -> 14 -> 15 -> 16 -> 17 -> *
F1
12 -> [13] -> 14 -> 15 -> 16 -> 17 -> *
R5
12 -> 17 -> 16 -> 15 -> 14 -> 13 -> [*]
I1
12 -> 17 -> 16 -> 15 -> 14 -> 13 -> 1 -> [*]
I9
12 -> 17 -> 16 -> 15 -> 14 -> 13 -> 1 -> 9 -> [*]
H1
12 -> [17] -> 16 -> 15 -> 14 -> 13 -> 1 -> 9 -> *
M5
12 -> 16 -> 15 -> 14 -> 13 -> 1 -> 17 -> [9] -> *
```

```
D1
12 -> 16 -> 15 -> 14 -> 13 -> 1 -> 17 -> [*]
H0
[12] -> 16 -> 15 -> 14 -> 13 -> 1 -> 17 -> *
I0
0 -> [12] -> 16 -> 15 -> 14 -> 13 -> 1 -> 17 -> *
```

Przy wykonywaniu programu nie wolno Ci przepisywać wartości z jednych węzłów listy do innych węzłów, możesz tylko zmieniać połączenia węzłów. Pamiętaj, żeby zwolnić instrukcją free ewentualne stworzone węzły pomocnicze, a także węzły usuwane komendą D. Twój program nie może nic pisać na standardowe wyjście (stdout), wypisywanie dowolnych rzeczy na standardowe wyjście błędów (stderr) jest dopuszczalne.

Zadanie

Należy dostarczyć plik z implementacją funkcji przetworz, używając pliku nagłówkowego lpro.h dostarczonego w pakiecie lpro_public.

Opis zawartości lpro_public (znajduje się też w pliku README.md):

lpro.h - plik nagłówkowy zawierający deklarację typu listowego (jak na zajęciach) oraz deklarację funkcji `przetworz`;

lpromain.c - plik zawierający funkcję `main`, która uruchamia funkcję `przetworz` na przykładzie wczytanym ze standardowego wejścia i wypisuje rezultat jej działania;

lpro_dummy.c - implementacja funkcji `przetworz`, która działa wyłącznie dla przykładu z treści zadania;

lpro_scheme.c - polecany schemat implementacji funkcji `przetworz`

lpro0a.in - plik wejściowy z prostym przykładem (patrz niżej);

lpro5a.in - plik wejściowy z przykładem z treści zadania (powyżej).

Aby uruchomić dostarczoną pseudo-implementację funkcji `przetworz` należy wydać polecenie

```
gcc -o lpro lpromain.c lpro_dummy.c
```

i następnie uruchomić powstały program wykonywalny

```
./lnad
```

Podobnie, gdy napiszesz własną wersję funkcji `przetworz` np. w pliku o nazwie lpro1.c należy go skompilować analogiczną komendą:

```
gcc -o lpro lpromain.c lpro1.c
```

i uruchomić jak wyżej.

Testowanie

Aby ułatwić Państwu stopniowe tworzenie rozwiązania, testy są ułożone następująco:

- test wstępny (z numerem 0) jest następujący:
  program `"H1I1I2"` na liście `10 -> 13 -> 14 -> *`
  spodziewany wynik to `11 -> 1 -> 2 -> 12 -> 13 -> *`
- testy z numerem 1 dotyczą wyłącznie komend F, H oraz I;
- testy z numerem 2 j.w. plus D;
- testy z numerem 3 j.w. plus M;
- testy z numerami 4 i więcej - cały zestaw komend;
- test z numerem 8 - test wydajnościowy - należy uważać z wydrukami (ale printList jest OK).