12 grudnia 2015

Zadanie 7

Proszę napisać (i przetestować) opisany niżej program:

Struktura opisująca węzeł listy ma postać

```
template <typename T>
struct Node {
    T     data;
    Node* next;
};
```

(każdy węzeł przechowuje dane typu **T**).

1. Napisać szablon funkcji o nagłówku

   ```
   Node<T>* arrayToList(const T tab[], size_t size);
   ```

   pobierającą tablicę i jej wymiar. Zadaniem funkcji jest utworzenie (używając oczywiście operatora **new**) listy jednokierunkowej obiektów struktury **Node<T>**, zawierającej (jako składowe **data**) w kolejnych węzłach kolejne elementy z przekazanej tablicy (w takiej samej kolejności). Funkcja ma zwracać wskaźnik do „głowy" utworzonej listy.
2. Napisać szablon funkcji o nagłówku

   ```
   Node<T>* extract(Node<T>*& head,
                    bool (*predicate)(const T&));
   ```

   pobierającą wskaźnik do „głowy" utworzonej listy. Funkcja ma wydzielić do osobnej listy te węzły, dla których funkcja (predykat) **predicate** zastosowana do pola **data** zwraca **true**. Funkcja **extract** zwraca wskaźnik do głowy (być może pustej) listy zawierającej węzły spełniające predykat, podczas gdy **head** po powrocie z funkcji zawiera wskaźnik do (być może pustej) głowy listy węzłów *niespełniających* predykatu (czyli **head** może być przez funkcję zmodyfikowane — dlatego jest przekazane przez referencję).
   UWAGA: funkcja **extract** operuje wyłącznie na *istniejących* węzłach typu **Node**, w żadnym przypadku nie tworzy nowych obiektów.
   Szablon **extract** może też mieć postać

   ```
   template<typename T, typename Pred>
   Node<T>* extract(Node<T>*& head, Pred predicate) {
       // ...
   }
   ```

   i wtedy możliwe będzie przy wywołaniu funkcji użycie zarówno wskaźnika funkcyjnego jak i lambdy.
3. Napisać szablon funkcji o nagłówku

```
void deleteList(Node<T>*& head);
```

zwalniającą (za pomocą **delete**) wszystkie węzły listy do której wskaźnik przekazany został jako argument. Przy każdym usuwaniu węzła funkcja powinna drukować wartość danej z usuwanego węzła, abyśmy widzieli, że rzeczywiście węzły te są usuwane. Po powrocie z funkcji head powinno być wskaźnikiem pustym (bo reprezentuje listę, która stała się pusta).

4. Napisać funkcję, także w postaci wzorca, która wypisuje elementy listy (w jednej linii, oddzielone spacją).

Przykładowy schemat programu:

```
#include <iostream>
#include <string>
using namespace std;

template <typename T>
struct Node {
    T   data;
    Node* next;
};

template <typename T>
void showList(const Node<T>* head);

template <typename T>
Node<T>* arrayToList(const T tab[], size_t size);

template<typename T>
Node<T>* extract(Node<T>*& head,
                 bool (*predicate)(const T&));

template <typename T>
void deleteList(Node<T>*& head);

bool isEven(const int& n)    { return n%2 == 0;     }
bool isLong(const string& s) { return s.size() >=5; }

int main() {
    int tabi[] = {2,1,4,3,6,5,7,8};
    size_t sizei = sizeof(tabi)/sizeof(tabi[0]);
    Node<int> *listAi = arrayToList(tabi, sizei);
    showList(listAi);
    Node<int> *listBi = extract(listAi,isEven);
    showList(listAi);
    showList(listBi);
    deleteList(listAi);
```

```
    deleteList(listBi);

    string tabs[] = {"Kasia", "Ola", "Ala",
                     "Zosia", "Ela", "Basia"};
    size_t sizes = sizeof(tabs)/sizeof(tabs[0]);
    Node<string> *listAs = arrayToList(tabs, sizes);
    showList(listAs);
    Node<string> *listBs = extract(listAs,isLong);
    showList(listAs);
    showList(listBs);
    deleteList(listAs);
    deleteList(listBs);
}
```

Program powinien wydrukować coś w rodzaju:

```
2 1 4 3 6 5 7 8
1 3 5 7
2 4 6 8
DEL 1; DEL 3; DEL 5; DEL 7;
DEL 2; DEL 4; DEL 6; DEL 8;
Kasia Ola Ala Zosia Ela Basia
Ola Ala Ela
Kasia Zosia Basia
DEL Ola; DEL Ala; DEL Ela;
DEL Kasia; DEL Zosia; DEL Basia;
```

Termin: do 27 grudnia (włącznie)

Rozwiązania, w postaci ***jednego*** *pliku źródłowego zawierającego treść programu, proszę wrzucać w systemie EDU do katalogu „Foldery zadań / Zadanie_XX", gdzie 'XX' jest numerem zadania.*

Nazwą pliku powinno być nazwisko z dużej litery (bez polskich znaków); rozszerzeniem musi być '`.cpp`*', czyli np.* `Malinowska.cpp`*.*