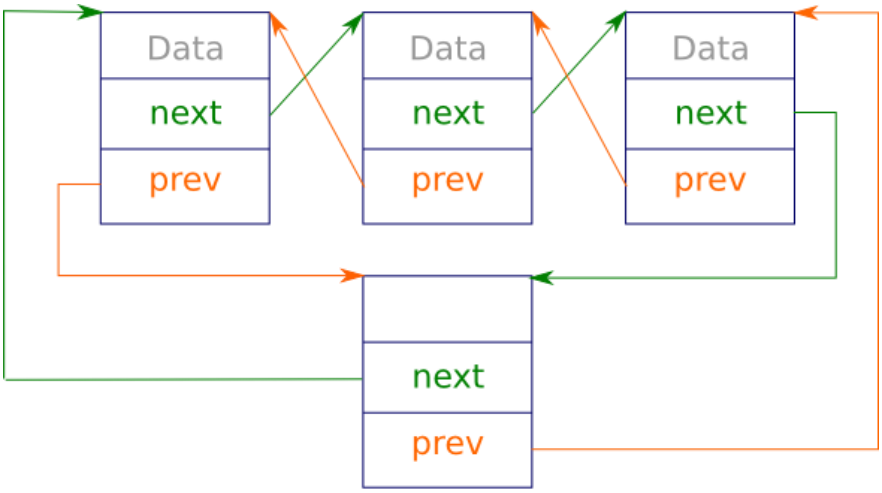
PJC 08

Zadanie 1

Lista dwukierunkowa (ang. doubly linked list) składa się z węzłów (ang. nodes), z których każdy zawiera dane pewnego typu oraz dwa wskaźniki: do węzła poprzedniego (prev) i następnego (next). Wygodna implementacja takiej listy polega na tym, że tworzymy węzeł-wartownika (ang. sentinel) z nieistotnymi danymi, którego składowa next wskazuje na pierwszy prawdziwy węzeł, a prev na ostatni, jak na rysunku



sentinel

Taka implementacja upraszcza operacje wstawiania i usuwania elementów listy.

Lista pusta reprezentowana jest przez wartownika, w którym oba składowe wskaźniki (prev i next) wskazują na niego samego.

Napisać (i przetestować) szablon klasy DLL, obiekty której reprezentują listy dwukierunkowe element w pewnego typu. Operacje określone dla listy definiowane są przez funkcje:

- konstruktor tworzący listę pustą;
- empty zwraca true jeśli lista jest pusta;
- push_front dodaje element na początek listy;
- push_back dodaje element na koniec listy;
- print_fwd wypisuje do zadanego strumienia wyjściowego (domyślnie jest to cout) wszystkie elementy listy (dane z poszczególnych węzłów, w jednym wierszu, oddzielone znakami odstępu);
- print_rev jak print_fwd, ale elementy wypisywane są od końca;
- find_first znajduje pierwszy węzeł, dla kt rego dana jest r wna podanej wartości i zwraca wskaźnik do tego węzła (co nieco pogwałca hermetyzację...) albo nullptr, jeśli takiego węzła nie ma;
- find_last jak find_first, tylko znajduje ostatni węzeł;

- insert_after otrzymuje wskaźnik do węzła (na przykład otrzymany wcześniej za pomocą nd_ rst) i wstawia węzeł o podanej danej za wskazanym;
- insert_before jak insert_after, ale wstawia nowy wezeł przed wskazanym;
- delete_node otrzymuje wskaźnik do węzła i usuwa go z listy; wypisuje dane z usuwanego węzła;
- reverse odwraca kolejność węzłów;
- clear usuwa wszystkie węzły listy (pozostawiając wartownika) tak, aby zmieniony obiekt reprezentował listę pustą; wypisuje dane z usuwanych węzłów, abyśmy mogli zobaczyć, że usuwane są wszystkie te węzły, które powinny zostać usunięte;
- destruktor, usuwający wszystkie węzły, łącznie z wartownikiem (może wywołać clear).

Program może mieć następującą strukturę:

```
 #include <iostream>
 #include <utility>     // swap (may be useful)
#include <string>

template<typenameT>
class DLL { struct Node
{
        T data;
         Node* next;
         Node* prev;
    };
Node* sent; // sentinel
public:
    DLL() : sent(new Node{T(),nullptr,nullptr}) { sent->next =
         sent->prev = sent;
    }

    bool empty() const; void
    push_front(const T& t) const; void
    push_back(const T& t) const;
    void print_fwd(std::ostream& str = std::cout) const; void
    print_rev(std::ostream& str = std::cout) const;
    Node* find_first(const T& e) const; Node*
    find_last(const T& e) const; void
    insert_after(Node* a, const T& t) const; void
    insert_before(Node* b, const T& t) const;
    void delete_node(const Node* d) const; void
    reverse() const; void clear() const;
    ~DLL();
};

int main () { using std::cout; using std::endl; using
    std::string;
```

```
    DLL<std::string>* listStr = new DLL<std::string>();
    listStr->push_back("X");
    listStr->push_back("E");
    listStr->push_front("C");
    listStr->push_front("X");
    listStr->push_front("A");
    cout << "List printed in both directions:" << endl;
    listStr->print_fwd();

    listStr->print_rev();

    listStr->delete_node(listStr-
    >find_first(łXł));

    listStr->delete_node(listStr->find_last(łXł));
    cout << "\nList after deleting X's:" << endl;
    listStr->print_fwd();

    listStr->insert_after(listStr->find_first("A"),"B");
    listStr->insert_before(listStr->find_last("E"),"D");
    cout << "List after inserting B and D:" << endl; listStr-
    >print_fwd();

    listStr->reverse();
    cout << "List after reversing:" << endl; listStr->print_fwd();

    std::cout << "Is list empty? " << std::boolalpha
              << listStr->empty() << std::endl; std::cout
    << "Clearing the list" << std::endl; listStr-
    >clear();

    std::cout << "Adding one element (Z)" << std::endl; listStr-
    >push_front("Z");

    std::cout << "Deleting the list" << std::endl;
    delete listStr;
}
```

Wszystkie metody są tu zadeklarowane jako const, bo żadna z nich nie zmienia formalnie stanu obiektu, którym jest adres węzła-wartownika. Powyższy program powinien wydrukować coś w rodzaju:

```
List printed in both directions:
A X C X E
E X C X A
del:X del:X
List after deleting X's:
A C E
List after inserting B and
D: A B C D E List after
```

```
reversing: E D C B A Is
list empty? false
Clearing the listt
DEL:E DEL:D DEL:C DEL:B DEL:A
Adding one element (Z)
Deleting the list
DEL:Z
```
