AiSD Projekt 2

Temat: Implementacja listy dwustronnie wiązanej

Wymagania organizacyjne:

- węzły listy są zadeklarowane pod nazwą `DLLNode`, sama lista powinna być zaimplementowana jako klasa o nazwie `DLL` i być umieszczona w przestrzeni nazw `AiSD`.

```
// DoublyLinkedList.h
...
namespace AiSD{

class DLLNode{
   //details
};

class DLL{
   //details
};
}//namespace AiSD
```

- Kod źródłowy musi być umieszczony w plikach `DoublyLinkedList.h/.cpp`
- Dopuszczalne jest umiesczenie klasy `DLLNode` jako klasy zagnieżdżonej w klasie `DLL`
- Implementacja powinna zachować interefejsy funkcji podane w wymaganiach, w przypadku modyfikacji należy uzasadnić powód.
- Jeżeli wymagania nie precyzują jakiejś kwestii oznacza to, że sposób jej realizacji jest pozostawiony do decyzji programisty.

Do implementacji musi być dołączone:

- sprawozdanie zawierajace:
  - wyliczenie spełnionych wymagań zgodnie z poniższą numeracją
  - do każdego spełnionego wymagania należy opisać gdzie i jak zostało ono spełnione, np.: wskazując miejsce w kodzie + opis
- program testujacy, demonstrujący każde spełnione wymaganie
  - w sprawozdaniu należy jasno wskazać w jaki sposób program testujący wykonuje demonstrację;
  - jeżeli jest to wygodne, można (trzeba) łączyć demonstracje dla różnych wymagań
  - w sprawozdaniu należy podać sposób uruchomienia, ewentualnej interakcji z użytkownikiem, tworzone pliki wyjściowe etc.
  - kod programu testujacego powinien być w pliku o nazwie: `projekt_2_<inicjaly_TL>_<inicjaly_>_demo.cpp`, np.: dla Adama Nowaka (TL) i Ewy Kowalskiej: `projekt_2_AN_EK_demo.cpp`

Wymagania podstawowe (muszą być spełnione)

1. W węzłach przechowywane są wartości typu `T=short`.

2. Dostępne są wszystkie wymienione niżej operacje. Zadbanie o to czy spełnione są warunki do wykonania operacji leży po stronie użytkownika, np.: usuwanie z listy lub odczytanie elementu nie musi sprawdzać czy lista jest pusta, jeżeli użytkownik nie sprawdzi tego sam przed wykonaniem operacji działanie programu jest nieokreślone.

    - `void PushFront(const T& el)`: wstawia element na początek listy; w czasie O(1)
    - `void PopFront()`: usuwa element z początku listy; w czasie O(1)
    - `void PushBack(const T& el)`: wstawia element na koniec listy; w czasie O(1)
    - `void PopBack()`: usuwa element z końca listy; w czasie O(1)
    - `T& Front()`: zwraca element z początku listy; w czasie O(1)
    - `T& Back()`: zwraca element z końca listy; w czasie O(1)
    - `bool IsEmpty()`: zwraca informacje czy lista jet pusta; w czasie O(1)
    - `size_t Size()`: zwraca ilość elementów na liście; w czasie O(n) lub O(1)
    - `void Clear()`: usuwająca wszystkie elementy listy, również fizycznie z pamięci; w czasie O(n)

3. Dostępne są funkcje wyświetlające zawartość listy na ekranie w kolejności ,,od początku'' i ,,od końca''.

4. Dostępna jest funkcja zapisująca zawartość listy do pliku.

Wymagania dodatkowe (mogą być spełnione)

1. Lista przechowuje złożony typ danych `T=Record`:

```
struct Record{
    std::string name;
    unsigned grade;
};
```

2. Klasa `DLLNode` udostepniaja:
    - konstruktor `DLLNode()`, który tworzy węzeł z next=prev=nullptr, data=T()
    - konstruktor `DLLNode(const DLLNode* n, const DLLNode* p, const T& t)`, który tworzy węzeł z `next=n`, `prev=p`, `data=t`
3. Klasa `DLL` udostępnia:
    - konstruktory domyślny, tworzący pustą listę
    - destruktor niszczący wszystkie elementy z listy; w czasie O(n)
4. Klasa `DLL` udostępnia operator `[]` pozwalający na odczyt i modyfikację i-tego elementu listy, niekoniecznie ze sprawdzaniem zakresu; w czasie O(n)
5. Dostępne są poniższe operacje:
    - `bool IsInList(const T& t)`: zwraca informację czy węzeł zawierający `t` jest na liście; w czasie O(n)
    - `DLLNode* Find(const T& t)`: zwraca wskaźnik do węzła zawierającego `t`, jeżeli nie ma na liście zwraca `nullptr`; w czasie O(n)
6. Dostępne są poniższe operacje:
    - `??? Insert(???)`: wstawia nowy element na wskazaną pozycję, sposób określenia pozycji zależny od implementacji; w czasie O(n)
    - `??? InsertAfter(???)`, `??? InsertBefore(???)`: wstawia element po/przed wskazanej pozycji, sposób określenia pozycji zależny od implementacji; w czasie O(n)
7. Dostępne są poniższe operacje:
    - `??? Delete(???)`: usuwa element ze wskazanej pozycji, sposób określenia pozycji zależny od implementacji; w czasie O(n); w czasie O(n)
    - `??? DeleteAfter(???)`, `??? DeleteBefore(???)`: usuwa element po/przed wskazaną pozycją, sposób określenia pozycji zależny od implementacji; w czasie O(n)
8. dotępna jest funkcja odczytująca listę z pliku; można zakładać, że dane w pliku są poprawnego typu i formatu, np.: przy wczytywaniu liczb bez znaku można zakładać, że plik nie zawiera liter lub liczb ze znakiem
9. Wszystkie operacje posiadają obsługę błędów: sprawdzają zakres indeksów, to czy lista jest pusta itp. Sposób obsługi błędów jest dowolny: mogą być ignorowane, ustawiana flaga, zwracana wartość logiczna lub wyrzucane wyjątki.