AiSD Projekt 1

Temat: Implementacja tablicy dynamicznej.

Wymagania organizacyjne:

- implementacja powinna być wykonana jako klasa o nazwie `DynamicArray` i być umieszczona w przestrzeni nazw `AiSD`.

```
// DynamicArray.h
...
namespace AiSD{

class DynamicArray{
   //details
};

}//namespace AiSD
```

- Kod źródłowy musi być umieszczony w plikach `DynamicArray.h/DynamicArray.cpp`
- Dopuszczalne jest wykonanie implementacji jako zbioru luźnych funkcji. W tym wypadku muszą byc one umieszczone w przestrzeni nazw `AiSD::DynamicArray`.
- 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_1_<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. Tablica przechowuje elementy typu T=long.
2. Rozmiar fizyczny tablicy jest stały, przechowywany przez stałą `size_t capacity`.
3. Rozmiar logiczny tablicy jest zmienny, przechowywany w zmiennej `size_t size`.
4. Indeksy elementów w tablicy należą do przedziału `[0,size-1]`
5. Dostępne są operacje wymienione niżej. Nie muszą sprawdzać zakresu indeksów ani dostępnego miejsca - zadbanie o sensowność operacji leży po stronie użytkownika; jeżeli podany został indeks z poza zakresu lub `size==capacity` zachowanie programu jest nieokreślone.
   a. `bool IsEmpty()`: zwraca `true` jeżeli tablica jest pusta, `false` w przeciwnym przypadku; w czasie O(1)
   b. `bool IsFull()`: zwraca `true` jeżeli tablica jest pełna, `false` w przeciwnym przypadku; w czasie O(1)
   c. `size_t Space()`: zwracająca ilość wolnego miejsca w tablicy; w czasie O(1),
   d. zapis/odczyt elementu tablicy o indeksie `i` w czasie O(1), zrealizowane jako (do wyboru):
      - `operator[]` działający tak jak wbudowany operator `[ ]` dla typu tablicowego,
      - para funkcji `T Get(size_t i)`/ `void Set(size_t i,T t)`: odczytująca/zapisująca wartość z/na pozycji `i`,
   e. `void PushBack(T t)`: wstawienie elementu `t` na koniec; w czasie O(1),
   f. `void PopBack()`: usunięcie elementu z końca; w czasie O(1),
   g. `void PushFront(T t)`: wstawienie elementu `t` na początek; w czasie O(size),
   h. `void PopFront()`: usunięcie elementu z poczatku; w czasie O(size),
   i. `void Insert(T t,size_t i)`: wstawia element `t` na pozycję `i`; w czasie O(size),
   j. `void Erase(size_t i)`: usunięcie elementu z pozycji `i`; w czasie O(size),
6. Dostępna jest funkcja wyświetlająca zawartość tablicy na ekranie wraz z informacją ilości wolnego miejsca.
7. Dostępna jest funkcja zapisująca zawartość tablicy do pliku wraz z informacją o ilości wolnego miejsca.

Wymagania dodatkowe (mogą być spełnione)

1. Rozmiar fizyczny jest zmienny, dynamicznie powiększany w przypadku przepełnienia.
2. Wszystkie operacje sprawdzają zakres indeksów i dostępne miejsce w tablicy. Sposób obsługi błędów jest dowolny: mogą być ignorowane, ustawiana flaga, zwracana wartość logiczna lub wyrzucane wyjątki. Wyjątkiem jest operator `[ ]`.
3. Wszystkie operacje gwarantują brak wycieków pamięci (no memory leaks).
4. Tablica przechowuje złożony typ danych T=`Record`:

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

5. Klasa `DynamicArray` udostępnia konstruktory:
   a. `DynamicArray(size_t capacity)`: tworzący pustą tablicę o pojemności `capacity>=0`
   b. `DynamicArray(size_t capacity, size_t N,const T& t)`: inicjalizuje tablicę za pomocą N kopii elementu `t`
6. Dostępne są operacje:
   a. `at(size_t i)`: funkcja realizująca dostęp do elementów tablicy ze sprawdzaniem zakresu indeksów; w czasie O(1),
   b. `void Clear()`: usuwa wszystkie elementy z tablicy; w czasie O(size)
   c. `size_t Search(const T& t)`: zwraca indeks elementu `t`, jeżeli elementu nie ma zwraca `size`; w czasie O(size)
7. Dostępne są operacje:
   a. `bool EraseFirst(const T& t)`: usuwa pierwszy element równy `t`; w czasie O(size)
   b. `size_t EraseAll(const T& t)`: usuwa wszystkie elementy równe `t`, zwraca liczbę usuniętych elementów; w czasie O(size)
   c. `size_t Erase(size_t from, size_t to)`: usuwa elementy o indeksach w przedziale `[from,to)`; w czasie O(size+(last-first))
8. Dotępna jest funkcja odczytująca zawartość tablicy z pliku. Plik może zawierać tylko dane, informacja o pojemności tablicy musi być określona na podstawie zawartości pliku. Można zakładać, że dane i format pliku są poprawne.
9. (*) Dostępna jest operacja `??? Insert(???)`: wstawia elementy `{t1,...,tk}` na pozycję `i`; w czasie O(size+k), typ oraz ilość argumentów zależny od implementacji
10. (*) Dostepne są: konstruktor inicjalizujący tablicę na podstawie listy elementów, copy-construktor, move-constructor, copy assignment operator, move assignment operator