Dokumentacja – „Tablica 2w”

- Zestawienie wykorzystywanych funkcji bibliotecznych

1. ```
   void* memcpy(void* destination, const void* source,
   size_t num);
   ```
   (z nagłówka <cstring>)

2. ```
   template<class T>
   void swap(T& a, T& b);
   ```
   (z nagłówka <algorithm>)

3. ```
   template <class RandomAccessIterator>
   void sort (RandomAccessIterator first, RandomAccessIterator last)
   ```
   (z nagłówka <algorithm>)

- Opis programu

Obsługa tablicy wielowymiarowej jest zaszyta w klasie szablonowej, która jako swój parametr przyjmuje nazwę typu **T**, który posłuży do zbudowania wskaźnika do tego właśnie typu **T** (czyli **T***) i przypisania mu adresu dynamicznie zaalokowanej tablicy elementów typu **T**.

Klasa wyposażona jest w konstruktor kopiujący i przeładowany operator przypisania, które umożliwiają bezpieczne kopiowanie obiektów tej klasy.

```
template <typename T>
class Tablica
{
    unsigned wiersze, kolumny;
    T* tab;

public:
    Tablica(const unsigned wier = 0, const unsigned kol = 0);
    Tablica(const Tablica& ob);
    Tablica& operator=(const Tablica& ob);
    ~Tablica();

    unsigned ile_wierszy() const;
    unsigned ile_kolumn() const;

    bool czytaj_z_pliku(const char* nazwa_pliku);
    void drukuj_na_ekran() const;
    void nadpisz_element(const unsigned wiersz, const unsigned kolumna,
    const T& wartosc);
    void dodaj_wiersz();
    void dodaj_kolumne();
    bool usun_wiersz(const unsigned wiersz);
    bool usun_kolumne(const unsigned kolumna);
    void zamien_elementy(const unsigned wiersz1, const unsigned kolumna1,
    const unsigned wiersz2, const unsigned kolumna2);
    void przestaw_kolumny(const unsigned kolumna1, const unsigned kolumna2);
    void przestaw_wiersze(const unsigned wiersz1, const unsigned
    wiersz2);
```
```

```
    void sortuj_tablice();
};
```

Kiedy obiekt jest niszczony, do pracy rusza jego destruktor, który dealokuje pamięć przydzieloną dla tablicy.

W funkcjach składowych klasy często wykorzystywana jest funkcja biblioteczna **memcpy()**, która kopiuje bajty pomiędzy zadanymi obszarami pamięci. Jest ona potencjalnie bardziej efektywna niż odpowiadająca jej pętla pisana przez programistę wykonująca to samo zadanie - stąd jej częste użycia.

W konstruktorze kopiującym i operatorze przypisania, przypisanie adresu nowo dynamicznie utworzonej tablicy do wskaźnika **tab** typu **T***, odbywa się na zasadzie:

```
tab = (ob.tab ? static_cast<T*>(memcpy(new T[wiersze*kolumny], ob.tab, wiersze*kolumny*sizeof(T))) : 0);
```

Jest tu użyty tzw. operator trójargumentowy. Jeśli obiekt wzorcowy **ob** (z którego kopiujemy zawartość) posiada zainicjalizowany wskaźnik **ob.tab**, to do wskaźnika **tab** obiektu, na rzecz którego wywołany został konstruktor kopiujący lub przeładowany operator przypisania, przypisujemy wartość:

```
static_cast<T*>(memcpy(new T[wiersze*kolumny], ob.tab, wiersze*kolumny*sizeof(T)))
```

Jest to właśnie wywołanie funkcji **memcpy()**, która kopiuje zwartość tablicy obiektu **ob** do nowo dynamicznie utworzonej tablicy, przesłanej jako pierwszy argument. Funkcja ta zwraca pierwszy argument wywołania, który rzutujemy za pomocą operatora rzutowania na typ **T***, ponieważ w C++ (w przeciwieństwie do C) rzutowanie z typu **void*** → **T*** trzeba przeprowadzić jawnie, za pomocą operatora.

Jeśli obiekt wzorcowy **ob** nie posiada zainicjalizowanego wskaźnika **ob.tab** (wskazuje on na adres zerowy), to do wskaźnika **tab** obiektu, na rzecz którego wywołany został konstruktor kopiujący lub przeładowany operator przypisania, przypisujemy również adres zerowy.

- Instrukcja obsługi programu

Program tworzy tablicę dwuwymiarową 5x6, a następnie wczytuje liczby całkowite z pliku „tablica.txt”, który musi znajdować się w tym samym folderze. Liczby oddzielone mogą być wyłącznie białymi znakami, a po ostatniej liczbie nie może być żadnego, nawet białego, znaku.

Program wczyta tyle liczb, na ile pozwala rozmiar tablicy, jeśli napotka wcześniej

jego koniec, resztę elementów tablicy zainicjalizuje (tak naprawdę wszystkie elementy są inicjalizowane podczas pracy konstruktora obiektu) wartością domyślną (w przypadku typu całkowitego **int** jest to wartość **0**).

Następnie program w funkcji **main()** wywołuje po kolei kolejne funkcje składowe klasy szablonowej aby zademonstrować działanie obiektu klasy imitującej tablicę.

Przy wywołaniach tych nie jest sprawdzana poprawność zakresu numerów wierszy i kolumn przekazywanych do funkcji składowych aby nie zaciemniać kodu źródłowego. W zwykłych przypadkach programista powinien zadbać o to, aby do funkcji tych zostały przekazywane poprawne wartości.