Lab_12

Sformułowanie zadania: obliczenie iloczynu skalarnego dwóch wektorów $dot = \mathbf{x}^T \cdot \mathbf{y} = \sum x_i \cdot y_i$; ($i \in [0, n-1]$). Wektory przedstawić w postaci tablicy struktur danych MY_VECTOR * *tab*, która zawiera 2 elementy: *tab*[0] – wektor **x**, *tab*[1] – wektor **y**. Użyć dynamiczne alokowanie pamięci.

```
struct MY_VECTOR
{
        int dim;            //rozmiar wektora
        char str[64];       //nazwa wektora
        double *buff;       //tablica dla przechowywania wektora o rozmiarze dim
};
```

Obsługa wektorów (pliki *.h, *.cpp) powinna zawierać:

Część minimalna:

1. Inicjowanie wektora.
2. Deinicjowanie wektora: zwolnienie pamięci dla tablicy buff.
3. Przygotowanie wektora: wprowadzenie rozmiaru zadania *n*, nazwy wektora *str* oraz alokowania pamięci i wypełnienia tablicy buff. Dla wektora **x**: $x_i = 1/(i+1)$, dla wektora **y**: $y_i = i+1$, $i \in [0, n-1]$.
4. Algorytmu mnożenia naiwnego: $dot = \sum_{i=0}^{n-1} x_i y_i$.
5. Sprawdzenie poprawności wyniku: dla podanych **x** i **y** $dot = n$.

Część zaawansowana:

6. Algorytmu mnożenia z ośmiokrotnym rozwijaniem pętli: $dot = \sum(x_i \cdot y_i + x_{i+1} \cdot y_{i+1} + \ldots + x_{i+7} \cdot y_{i+7})$; indeks *i* zmienia się w pętle z przyrostem 8. Jeśli rozmiar zadania *n* nie jest wielokrotny liczbie 8, wydzielamy główną część zadania $n_1 \% 8 = 0$ ($n_1 = n - n\%8$), rozwijamy pętle dla n_1, a pozostałą resztę *n%8* obliczamy w sposób naiwny: $dot = \sum(x_i \cdot y_i + x_{i+1} \cdot y_{i+1} + \ldots + x_{i+7} \cdot y_{i+7}) | i \in [0, n_1 - 1] + \sum x_i \cdot y_i | i \in [n_1, n-1]$.
7. Wykonać pomiary czasu, używając funkcje DWORD GetTickCount() (dołożyć #include “windows.h”), i porównać wyniki dwóch algorytmów. Rozmiar zadania przyjąć 100 000 000. Projekt uruchomić na platformie x64.

Na monitor wyprowadzić czas obliczeń i OK, jeśli wyniki są poprawne, albo *error*!!!!!!!! w przypadku błędnych wyników.

Funkcja main() powinna zawierać:

1. Tworzenie tablicy MY_VECTOR * *tab*.
2. Inicjowanie i wypełnienie elementów tablicy (wektora **x** oraz wektora **y**).
3. Wykonanie algorytmu mnożenia naiwnego przy użyciu pomiarów czasu.
4. Sprawdzenie poprawności wyniku.
5. Wykonanie algorytmu mnożenia zaawansowanego przy użyciu pomiarów czasu.
6. Sprawdzenie poprawności wyniku.
7. Deinicjowanie wektorów **x**, **y**.
8. Zwolnienie pamięci dla tablicy *tab*.

W funkcji *main* nie wykonujemy żadnej arytmetyki (za wyjątkiem pomiarów czasu), tylko wywołujemy funkcje z obsługi wektora. Obsługę błędów i komunikatów umieszczamy do plików obsługi (*.h, *.cpp).

Stworzyć funkcje GLOBAL_Dealloc(), która powinna zwolnić poprawnie pamięć, alokowaną dynamicznie. Użyć tą funkcje w obsłudze błędów i komunikatów przy wystąpieniu błędów fatalnych.

Pomiary czasu:

DWORD ts = GetTickCount();
//Fragment kodu, dla którego wykonujemy pomiary czasu
double elapsed_time = ((*double*)*GetTickCount*() – *ts*); //czas w milisekundach.