ZADANIA DO SAMODZIELNEGO ĆWICZENIA

Ćwiczenie 0

Napisz program, który

1. Utworzy na stosie obiekt $std :: vector < double >$.
2. Wypełni go 7 razy liczbą 21,37.
3. Wypisze go na standardowe wyjście z użyciem iteratorów.
4. Zwolni pamięć w poprawny sposób.

Ćwiczenie 1

Zaimplementuj poniższe dwie funkcje

$$f(x,y) = \frac{1}{x^2 - y^2} \tag{1}$$

$$g(z) = e^z \tag{2}$$

Pierwsza funkcja powinna zwracać NAN w przypadku próby podzielenia przez zero. Druga funkcja powinna zwracać -1, jeżeli z jest NAN. Użyj poniższego przykładu:

```
#include <iostream>
#include <cmath>

double f(double x, double y);
double g(double z);

int main()
{
        std::cout << g( f( 2, -1) ) << std::endl;
        std::cout << g( f( 3, -3) ) << std::endl;
        return 0;
}
```

Output:
1.39561
-1

Ćwiczenie 2

Napisz funkcję

```
std::vector<unsigned>* transform(unsigned* a, unsigned *b, unsigned* c)
```

która przyjmuje wskaźniki na 3 elementy tej samej tablicy, przy czym a wskazuje na wcześniejszy element niż b, b wskazuje na wcześniejszy element niż c. Funkcja ma za zadanie stworzyć na stosie wektor, wypełnić go najpierw parzystymi liczbami na pozycjach między a i b włącznie, a potem nieparzystymi liczbami na pozycjach od b+1 do c włącznie. Na koniec funkcja ma zwrócić wskaźnik na utworzony wektor. Przetestuj dla sensownego przykładu, co najmniej 20 elementowej tablicy, na przykład kolejnych liczb naturalnych.

Ćwiczenie 3

Napisz funkcję

```
void foo(unsigned* a, unsigned b*)
```

która dla elementów tablicy od a do b włącznie zastosuje nastepujący algorytm: Jeżeli liczba jest podzielna przez 3 to zamień ją na 3, w przeciwnym razie na 1, chyba, że jest zerem, wtedy jej nie zmieniaj. Przetestuj dla sensownego przykładu. Przykład:

```
#include <iostream>

using namespace std;

void foo(unsigned* a, unsigned*  b);

int main()
{
        unsigned tab[10] = {0, 2, 3, 6, 23, 27, 9, 0, 1, 33};
        foo(tab, &tab[9]);
        cout << "[ ";
        for(auto i : tab)
                cout << i << " ";
        cout << "]" << endl;
        return 0;
}
```

Output:
[0 1 3 3 1 3 3 0 1 3]

Ćwiczenie 4

Uzupełnij definicję pozniższej funkcji

```
void odwroc(double * pa, double * pb)
```

która dostaje wskaźniki do dwóch elementów z tablicy double i ma za zadanie zmodyfikować tablicę tak, żeby kolejność wszystkich elementów stojących pomiędzy elementami wskazywanymi przez pa i pb (łącznie z tymi wskazywanymi) została odwrócona, tzn. elementy *pa i *pb zamieniamy miejscami, elementy *(pa+1) i *(pb-1)[1] zamieniamy ze sobą itd. Załóż, że wskaźniki są niepuste, wskazują elementy z tablicy zaalokowanej jako jeden blok w pamięci. Nie zakładaj, że element wskazywany przez pa stoi wcześniej w tablicy niż ten wskazywany przez pb. Nie twórz żadnych nowych tablic ani kolekcji.

Ćwiczenie 5

Napisz program, który poprosi użytkownika o podanie liczby całkowitej dodatniej, która będzie rozmiarem tablicy. Następnie program w zewnętrznej funkcji utworzy na stosie (dynamicznie) tablicę o podanym rozmiarze, wypełni ją kolejnymi liczbami pierwszymi poczynając od 2 i zwróci do funkcji main odpowiedni wskaźnik na tablicę. Na koniec program wyświetli elementy tablicy na standardowe wyjście i zwolni zaalokowaną pamięć.

[1] O ile pa wskazuje element wcześniejszy.