

Задачи по теме "Строки, Шаблоны и STL"

Часть 1: Строки в C++

```
#include <iostream>
#include <string>
using std::cout;
using std::endl;

int main ()
{
    /* В языке C++ есть библиотека string, которая предоставляет класс строк
     * std::string.
     * В отличие от строк языка C (char*) строки std::string в языке C++
     * гораздо проще и удобней.
     */
    std::string a = "Deus";

    // Можно присваивать ( В языке C пришлось бы использовать strcpy )
    std::string b;
    b = "machina";

    // Можно складывать ( В языке C пришлось бы использовать strcat )
    std::string c = a + " ex " + b;
    cout << c << endl;

    // Можно сравнивать ( В языке C пришлось бы использовать strcmp )
    if (b > a)
        cout << "String b is greater than string a" << endl;
}
```

1. **String 1:** Создайте строку "Hello world" и напечатайте её на экран.
2. **String 2:** Создайте программу, которая будет считывать слова (используете cin) в бесконечном цикле и каждый раз печатать сумму всех слов. Например, если пользователь ввёл Hello, то программа должна напечатать Hello и запросить следующее слово. Если затем пользователь введёт World, то программа должна будет напечатать HelloWorld и запросить следующее слово.

Часть 2: Шаблонные функции

```
#include <iostream>
#include <string>
using std::cout;
using std::endl;

template <class T>
T GetMax (T a, T b)
{
    if (a > b)
        return a;
    else
        return b;
}

int main ()
{
    int a = 5, b = 6;
    cout << GetMax<int>(a, b) << endl;

    long long n = 9645634567, m = 7356735634;
    cout << GetMax<long long>(n, m) << endl;

    float x = 4.634, y = 534.346;
    cout << GetMax<float>(x, y) << endl;

    std::string s1 = "deus", s2 = "machina";
    cout << GetMax<std::string>(s1, s2) << endl;
}
```

1. **Шаблоны 1:** Написать шаблонную функцию `T triple(T x)`, которая увеличивает переменную в 3 раза. Проверить её на переменных типа `int`, `float`, `Complex`, `std::string`.

```
cout << triple<int>(5) << endl;
// Должно напечатать 15

std::string s = "Hello"
cout << triple<std::string>(s) << endl;
// Должно напечатать HelloHelloHello
```

2. **Шаблоны 2:** Написать шаблонную функцию `T sum(T arr[], int count)`, которая возвращает сумму массива переменных. Проверить её на переменных типа `int`, `float`, `Complex`, `std::string`.

```
int numbers[] = {4, 8, 15, 16, 23, 42};
cout << sum<int>(numbers, 6) << endl;
// Должно напечатать 108

std::string words[] = {"Deus", "Ex", "Machina"};
cout << sum<std::string>(words, 3) << endl;
// Должно напечатать DeusExMachina
```

Часть 3: std::vector

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using std::cout;
using std::endl;

int main ()
{
    // std::vector - это удобный динамический массив C++
    // Он хранит все элементы в куче и автоматически увеличивается в размерах, если нужно
    std::vector<int> v = {54, 62, 12, 97, 41, 6, 73};
    cout << v[1] << endl;
    // Напечатает 62

    // Для добавления новых элементов в вектор используйте push_back
    v.push_back(44);

    // Можно узнать размер вектора и его вместимость(capacity)
    cout << "Size = " << v.size() << "    Capacity = " << v.capacity() << endl;

    // Для того, чтобы пройти по всему вектору:
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
    cout << endl;

    // Другой способ - использование итераторов:
    // Итератор это специальный объект созданный для удобства обхода структуры данных
    // v.begin() - это итератор, который указывает на первый элемент
    // v.end() - это итератор, который указывает на элемент следующий за последним
    // *it - получить сам элемент по итератору
    // В случаи вектора итераторы во многом похожи на обычные указатели
    // Различия проявляются в случаи более сложных структур данных например( деревьяев)
    for (std::vector<int>::iterator it = v.begin(); it != v.end(); ++it)
        std::cout << *it << ' ';
    cout << endl;

    // Для сортировки вектора можно использовать стандартную функцию std::sort из
    // библиотеки algorithm
    std::sort(v.begin(), v.end());

    // Для поиска элемента в векторе нужно использовать стандартную функцию std::find из
    // библиотеки algorithm. Эта функция вернёт итератор на элемент. Если элемента
    // в векторе нет, то функция вернёт v.end()

    if (std::find(v.begin(), v.end(), 55) != v.end())
        cout << "Element found" << endl;
    else
        cout << "Element not found" << endl;
}
```

1. **Размер и вместимость:** Проверьте как работает автоматическое расширение вектора. Для этого создайте пустой вектор и заполните его числами от 0 до 300 (используйте `push_back`). При этом на каждом шаге печатайте размер вектора и его вместимость.
2. **Reserve:** Постоянные расширения вектора могут быть очень трудозатратны. Используйте метод `reserve`, чтобы расширить вектор до значения 300 перед добавлением элементов. Проверьте как будет меняться размер и вместимость вектора в этом случае.
3. **Обратить вектор:** Используйте функцию `reverse` из библиотеки `algorithm`, чтобы обратить вектор `v`.
4. **Вектор в функции:** Написать функцию `square_vec`, которая принимает на вход вектор чисел типа `int` и возводит все числа вектора в квадрат.
5. **Вектор строк:** Создадим следующий вектор строк:

```
std::vector<std::string> animals = {"Cat", "Dog", "Bison", "Dolphin", "Eagle", "Pony",  
    "Ape", "Lobster", "Monkey", "Cow", "Deer", "Duck", "Rabbit", "Spider", "Wolf",  
    "Turkey", "Lion", "Pig", "Snake", "Shark", "Bird", "Fish", "Chicken", "Horse"};
```

- Напечатайте это вектор на экран.
 - Отсортируйте этот вектор и напечатайте его.
 - Обратите этот вектор и напечатайте его.
 - Напечатайте только тех животных, которые начинаются на букву S.
 - Написать функцию `get_first_letter`, которая принимает на вход вектор строк и один символ. Эта функция должна должна возвращать вектор строки слов, которые начинаются на соответствующий символ. Для этого внутри функции вы должны создать новый вектор, заполнить его нужными строками и вернуть. Проверить правильность работы функции, вызвав её в функции `main` и напечатав результат.
6. **Шаблоны + векторы:** Написать шаблонную функцию `T sum(std::vector<T> vec)`, которая возвращает сумму массива переменных. Проверить её на переменных типа `int`, `float`, `Complex`, `std::string`.
 7. **Is exist:** Напишите функцию `is_exists`, которая будет проверять есть ли в векторе элемент `x`.

Часть 4: std::set

```
#include <iostream>
#include <string>
#include <set>
#include <algorithm>
using std::cout;
using std::endl;

int main ()
{
    // Множество это контейнер для быстрого добавления, удаления и поиска
    // Под капотом это бинарное дерево поиска с балансировкой
    // Соответственно, все операции выполняются за O(n)
    std::set<int> s = {54, 62, 12, 97, 41, 6, 73};
    // Доступ по индексу не работает
    // cout << s[1] << endl;

    // Для добавления новых элементов в вектор используйте insert
    s.insert(44);

    // Для поиска элементов используйте find
    // Функция find возвращает итератор
    // Если элемента в множестве нет, то функция вернёт s.end()
    std::set<int>::iterator it = s.find(20);
    if (it != s.end())
        cout << "Element is found" << endl;
    else
        cout << "Element isn't found" << endl;

    // Для удаления используйте функцию erase
    s.erase (s.find(41));

    // Для прохода по множеству используйте итератор
    for (std::set<int>::iterator it = s.begin(); it != s.end(); ++it)
        cout << *it << " ";
    cout << endl;
    // Обратите внимание, что множество всегда отсортировано
    // Независимо от того как вы положили туда элементы
    // Это работает так как set является бинарным деревом поиска
}
```

1. Множество строк: Дано следующее множество строк

```
std::set<std::string> animals = {"Cat", "Dog", "Bison", "Dolphin", "Eagle", "Pony",
    "Ape", "Lobster", "Monkey", "Cow", "Deer", "Duck", "Rabbit", "Spider", "Wolf",
    "Turkey", "Lion", "Pig", "Snake", "Shark", "Bird", "Fish", "Chicken", "Horse"};
```

- Добавьте в множество новый элемент “Hippo”.
- Напечатайте все элементы множества.
- Удалите строку “Monkey”

2. Vector скорость работы: Создайте vector размера 10^7 и заполните его случайными числами от 0 до $2 * 10^7$. Попробуйте найти в этом векторе число 777. Сгенерировать случайное число можно так.

```
#include <stdlib.h>
#include <time.h>
int main()
{
    srand(time(0));
    int a = rand() % 2e7;
}
```

Создайте set размера 10^7 и заполните его случайными числами от 0 до $2 * 10^7$. Попробуйте найти в этом векторе число 777. Сравните поиска элемента вектора и множества.