//Практика 1:

// enum class

// initializer\_list и универсальные списки инициализации

// auto

// decltype

// lambda

// template variable

int main()

{

//Задание 1.

{

enum months { January, February, March/\*...\*/ };

enum weekDays { Monday, Tuesday /\*...\*/ };

months m = January;

//а) уберите проблемы (это значит, что КОМПИЛЯТОР не

//должен позволять программисту осуществлять опасные

//сравнения!)

if (m == Monday) { /\*...\*/ }

//б) оптимизируйте использование памяти

months year[] = { January, February, March };

size\_t n = sizeof(year); //???

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 2. Создайте и с помощью списка инициализации

//заполните значениями вектор с элементами - string

//С помощью range-based for преобразуйте все буквы в

//верхний регистр а остальные символы должны остаться

//неизменными

{

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

///Задание 3. Создайте и заполните значениями

//map двумя способами (3а и 3б) таким образом, чтобы он содержал

//в качестве ключа букву, а в качестве значения

//отсортированную по алфавиту совокупность слов (string),

//начинающихся с этой буквы

//3а. Проинициализируйте map "вручную" посредством списка инициализации

//С помощью range-based for и structured binding

//распечатайте содержимое, например: A: any, apple, away

{

\_\_asm nop

}

//3б. Создайте пустой map и используйте заданный массив

//для выполнения задания.

//С помощью range-based for и structured binding

//распечатайте содержимое, например: A: any, apple, away

{

//дано (например):

const char\* s[] = { "yong", "away", "bar", "any", "son", "apple" };

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 4. создать функцию для вывода на печать

//элементов последовательностей, заданных ниже:

{

std::vector<double> vd = { 1.1,2.2,3.3 };

//PrintAnyCont(vd);

std::string s("abc");

//PrintAnyCont(s);

int ar[] = { 1, 2, 3 };

//PrintAnyCont(ar);

std::initializer\_list<int> il{ 3,4,5 };

//PrintAnyCont(il);

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

///Задание 5.

//Cоздать функцию для "отрицания" значений, например:

//было: {1, -2, 5}, стало: {-1, 2, -5})

//изменение объектов типа std::string может выглядеть "aBc1" -> "AbC1"

//элементов последовательностей, заданных ниже:

{

std::vector<double> vd{ 1.1,2.2,3.3 };

//NegateAll(vd);

std::list<std::string> ls{ "aBc", "Qwerty", "n12" };

//NegateAll(ls);

int ar[]{ 1, 2, 3 };

//NegateAll(ar);

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 6. Реализовать функцию сортировки по модулю

//элементов последовательностей, заданных ниже

//Собственно для сортировки можно использовать обобщенный

//алгоритм sort(), а для задания условия - лямбда-функцию

{

std::vector<double> vd = { -3.3, 2.2, -1.1 };

//absSort(vd);

int ar[] = { -3, 2, -1 };

//absSort(ar);

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 7.

//Напишите функцию, которая будет формировать и

//возвращать вектор, каждый элемент

//которого является суммой элементов двух

//последовательностей РАЗНОЙ длины

//и с элементами РАЗНОГО типа.

//Подсказка 1: так как последовательности могут быть

//разной длины, логично сделать размер результирующего

//вектора максимальным из двух

//Подсказка 2: подумайте о возможности использования

//алгоритмов copy() и transform(), в котором

//трансформирующее действие удобно в свою очередь

//задавать лямбда-функцией

//например:

{

std::vector<int> v{ 1,2,3,4 };

std::list<double> l{ 1.1, 2.2, 3.3, 4.4, 5.5 };

//??? = SumCont(v, l);

std::list<int> ll{ 1, 2, 3, 4, 5, 6, 7, 8 };

double ar[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };

//??? = SumCont(ar, ll);

std::set<std::string> s{ "abc", "qwerty", "my"};

std::deque<const char\*> d{ "111", "22" };

//??? = SumCont(s, d);

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 8. Реализуйте функцию, которая принимает следующие параметры:

//сформированную последовательность любого типа с элементами любого типа,

//два (пустых) контейнера любого типа из vector, list, deque, set

//с элементами того же типа, что и у сформированной последовательности

//Функция должна "разложить" значения заданной последовательности в два пустых контейнера

//согласно заданному условию. Условие задать лябда-функцией

//Исходная последовательность при этом не меняется

{

//Например:

std::vector<int> v{ 1,2,3,4,5 };

std::list<int> l; //сюда четные

std::deque<int> d; //а сюда нечетные

//Separate(v, l, d, <условие>);

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 9. C помощью алгоритма for\_each()!!!

//(а не count\_if()) посчитать сколько букв в верхнем

//регистре.

// Использовать лямбда функцию

{

char s[] = "Hello World!";

//for\_each

\_\_asm nop

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Задание 10. Реализовать конвертацию enum в строковое представление - enumToString

// и наоборот - stringToEnum

//Подсказки:

//\*\*\*\*\*\*\*\*\*\*\*

//1. Соответствующие именованным константам строки все равно нужно где-то хранить =>

//1.1 Именованные константы в перечислении должны быть уникальными => соответствующие строки

//тоже должны быть уникальными, => уникальные строки могут быть использованы в качестве

//ключа в std::map

//1.2 а вот значения (соответствующие именованым константам)

//могут быть любыми и даже могут повторяться (упрощаем задачу, считая, что значения НЕ повторяются)

//=> подходит контейнер std::map<string,<значение> >

//1.3 Согласно стандарту С++11 переменные перечислимого типа могут быть разного размера и типа,

//а хотелось бы обеспечить универсальную конвертацию

//\*\*\*\*\*\*\*\*\*\*\*

//2.1 Так как типы перечислений разные, то enumToString и stringToEnum должны быть шаблонными

//2.2 Пользователь может ошибиться или злонамеренно передать в функции значения, которым

// ничего не соответствует => защита!

//\*\*\*\*\*\*\*\*\*\*\*

//3. Чтобы действия с map<string, <значение> > не зависили от типа перечисления, подумайте над использованием

//шаблонной переменной (в частности вспомните о возможности специялизации шаблонной переменной)

//\*\*\*\*\*\*\*\*\*\*\*

{

//Например:

/\*

COLORS c1;

try {

c1 = stringToEnum<COLORS>("blue");

}

catch (...)

{

//...

}

auto Str = enumToString(c1);

\*/

\_\_asm nop

}

return 0;

}