#include <string>

#include <iostream>

#include <cstdint>

#include <algorithm>

#include <iterator>

#include <memory>

using namespace std;

int main()

{

//Задание 1. Сырые строковые литералы (Raw String Literals)

//Выведите строку, например: my name is "Marina"

//а) без использования Raw String Literals

//б) посредством Raw String Literals

//в) добавьте разделители (Delimeter)

{

\_\_asm nop

}

//////////////////////////////////////////////////////////////////////////////////////////////

//Задание 2. Реализуйте вычисление факториала с помощью constexpr-функции.

//

//Подсказки/напоминания:

// - constexpr – функция должна состоять из единственной инструкции return <выражение>; (пока!)

// - но это выражение может включать вызов другой constexpr – функции, в частности рекурсивный

// вызов

// - если параметр рекурсивной constexpr- функции - это константа, компилятор вычислит результат

// на этапе компиляции

//Проверьте тот факт, что компилятор вычисляет значение на этапе компиляции (если в качестве

// параметра используется константа, известная компилятору на этапе компиляции).

// Для проверки достаточно создать встроенный массив с размерностью, вычисляемой

// посредством constexpr-функции:

{ //Например:

//int ar[factorial(3)];

//или

//constexpr int n = factorial(5);

//int ar1[n];

//попробуйте:

//int m = 7;

//constexpr int n1 = factorial(m);

//int ar1[n1];

//а так?

//int n2 = factorial(m);

\_\_asm nop

}

//////////////////////////////////////////////////////////////////////////////////////////////

//Задание 3a. Перевод с помощью пользовательского литерала из двоичного представления строкового

//в значение, например: строку "100000000" -> в значение 256

//Проверить результат посредством префикса 0b

//Чтобы вызов пользовательского литерала выглядел просто и читаемо, например: 100000000\_b

//логично использовать пользовательский литерал с единственным параметром - const char\*

//Так как речь идет о литералах, логично вычислять значения на этапе компиляции

// => реализуйте пользовательский литерал посредством constexpr - функций

//Подсказка/напоминание:

// - constexpr – функция должна состоять из единственной инструкции return <выражение>;

// - но это выражение может включать вызов другой constexpr – функции,

// - которая может быть рекурсивной (если параметр такой функции - это константа,

// компилятор вычислит результат вызова рекурсивной функции на этапе компиляции)

{

\_\_asm nop

}

//Задание 3b. Перевод в строковое двоичное представление, например: 256 -> "0b100000000"

//Так как строка может быть любой длины, логичнее и проще возвращать объект std::string

//=> возвращаемое значение не может быть constexpr!

//Подсказка: манипулятора std::bin пока нет => преобразование в двоичную строку

//придется делать вручную

//Подсказка: количество разрядов в байте определяет константа CHAR\_BIT - <cstdint>

{

//std::string sBin= 256\_toBinStr;

\_\_asm nop

}

//////////////////////////////////////////////////////////////////////////////////////////////

//Задание 4а. constexpr - объекты

//Создать класс (шаблон класса?) для хранения и манипулирования диапазоном значений.

//В классе должны быть:

// переменные для хранения минимального и максимального значений,

// методы для получения каждого из значений

// метод для проверки - попадает ли указанное значение в диапазон

// метод, который получает любое значение данного типа и формирует результирующее значение:

// если принадлежит диапазону, то его и возвращаем

// если меньше минимального значения, возвращаем минимальное

// если больше максимального значения, возвращаем максимальное

//Проверьте тот факт, что компилятор вычисляет значение на этапе компиляции.

{

\_\_asm nop

}

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

//Задание 5.

/\*

Реализуйте шаблон функции для печати любых последовательностей (vector, list, deque, set и встроенного массива), которые могут содержать:

• как объекты любого типа,

• так и указатели на объекты любого типа (указатели распечатывать неинтересно => в этом случае следует получать значение по адресу)

Подсказки: if constexpr

\*/

{

}

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

//Задание 5.

/\* Реализуйте шаблон функции сложения двух значений.

Если первое слагаемое является вектором, то все элементы вектора нужно увеличить на значение второго параметра. При этом элементы вектора и второй параметр должны быть одного и того же типа.

Подсказки: if constexpr, is\_same

\*/

{

}

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

//Задание 7.

/\* Реализуйте шаблон функции вывода на печать значений элементов любого адаптера (stack, queue, priority\_queue)

Подсказки: if constexpr, is\_same

Предусмотрите вывод значений, если в адаптере хранятся указатели.

\*/

{

}

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

//Задание 8.

/\* Реализуйте шаблон constexpr функции Smth(), которая должна возвращать значения разного типа

Подсказки: constexpr, if constexpr

\*/

//constexpr int res1 = /\*<вызов Smth()>;\*/ //res1 = 1

//constexpr double res2 = /\*<вызов Smth()>; \*/ //res2 = 2.2

// /\*constexpr???\*/ std::string res3 = /\*<вызов Smth()>; \*/ //res3 = "abc"

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

//Задание 9.

/\*Пользовательский deduction guide – для вывода типов параметров шаблона

Задан шаблон класса, который инкапсулирует внедренный ограниченный массив известной

размерности с элементами любого типа. \*/

/\*

template<typename T, size\_t size> class MyArray

{

T ar[size]; //как обеспечить инициализацию элементов базового типа по умолчанию нулем?

…

public:

MyArray(const T\*[, возможно другие параметры]);

};

\*/

/\*

//Требуется обеспечить работоспособность приведенных примеров использования.

{

MyArray<int, 5> ar1;//MyArray<int,5>

MyArray<char, 5> ar2{"ABCqwerty"};//MyArray<char,5>

MyArray ar3{"ABC"}; //MyArray<char,4>

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

MyArray ar4{ ar };

}

\*/

}