«В рамках вакансии Software developer (Lua) предлагаем выполнить тестовое задание:

Требуется написать программу управления сборочным конвейером.

Конвейер состоит из движущейся ленты, на которой установлены детали, и исполнительных механизмов (далее ИМ для краткости).
Конвейер разбит на участки, каждый из которых обслуживает отдельный ИМ. Технологический цикл работы конвейера включает в себя сдвиг ленты и обработку деталей.
Сдвиг ленты возможен, только если все ИМ сообщили об успешном выполнении операции. Если ИМ сообщает об аварии или не отвечает в течение заданного времени, то конвейер останавливается и регистрируется авария, после чего возврат в автоматический режим работы возможен только по команде оператора. После сдвига ленты, ИМ по команде управляющей программы выполняет одну технологическую операцию над деталью. После того как все ИМ успешно отработали операцию, технологический цикл повторяется.

Программу можно написать на любом знакомом языке или псевдокоде.»

«Ответили:
управляющую программу ИМ реализовывать не нужно, но если он захочешь написать ее имитацию с целью демонстрации работоспособности алгоритма управления конвейером, мы возражать не будем.»

В программе реализовано два класса.

Первый класс Conveyor (конвейер) содержит в себе:

- uint16_t number_implementers; //количество ИМ в конвейере
- uint16_t number_details; //количество Деталей в конвейере
- uint16_t step; //Номер Шага (цикла) конвейера
- std::unique_ptr<Implementer[]> ptr = nullptr; //дин массив для ИМ

и имеет методы:

- конструктор без параметров с пользовательским вводом
- конструктор с параметрами с вызовом исключений
- void static repair() – ремонтирует отдельный ИМ
- uint16_t crash() const – вызывает оператора для определения действия починить ИМ или завершить работу конвейера
- void process (int16_t const input_amount_steps) – метод реализующий работу конвейера. Определение «занятости» каждого ИМ. Получения результата работы каждого ИМ, и переход конвейера на следующий шаг (сдвиг ленты), если все ИМ дали положительный отклик или были починены посредством вызова функций вызова оператора crash() и вызова ремонта repair()
- uint16_t get_step() const – получение значения шага конвейера
- void show() const – вывод характеристики конвейера

Второй класс Implementer (ИМ) содержит в себе:

- int state; //состояние ИМ

  state будет принимать значения из:

```
enum states{            //состояния для ИМ
   wait,                //0  ИМ ожидает деталь
   success,             //1  ИМ успешно отработал с деталью
   error = -1,          //-1 ИМ получил ошибку, работая с деталью
   no_answer = -2       //-2 ИП получил проблему "не отвечает в течение заданного
времени", работая с деталью
};
```

- bool busy; //показатель "работает ли на текущем шагу (цикле конвейера) данный ИМ" если значение busy равно 1, то данный ИМ будет на этом цикле(шаге) работать с деталью.

и имеет методы:

- конструктор по умолчанию
- int get_state() const – получение состояния ИМ
- void set_busy(bool input_busy) – установка значения «занятости» отдельного ИМ
- void show() – вывод результата работы над деталью отдельного ИМ
- int technological_operation() – определяет с помощью вероятностей и рандомизации, каков будет результат работы ИМ над деталью

Для удобства прогона программы под различные характеристики конвейера можно воспользоваться дополнительным конструктором, изначально находящегося закомментированным.

```
int main() {
    system( _Command: "chcp 65001");
    int16_t amount_steps = 0;
    std::cout << "How many steps should the pipeline perform?" << std::endl;
    std::cin >> amount_steps;
    try {
        //Conveyor a(10, 4); //Конструктор с параметрами и throw
        Conveyor a;         //Конструктор с пользовательским вводом
        a.process(amount_steps);
    }
    catch (std::exception const &e) {
        std::cerr << e.what() << '\n';
    }
    catch (...) {
        std::cout << "Unknown error" << '\n';
    }
    return 0;
}
```

Разбор вывода состояния конвейера и его ИМ

```
Number of implementers = 5
Number of details = 10
 0:  0  0  0  0  0  S = 5
 1:  1  0  0  0  0  S = 5
 2:  1  1  0  0  0  S = 5
 3:  1  1 -1  0  0  S = 4
```

Number of implementers = 5 – количество ИМ в конвейере

Number of details = 10 – количество деталей необходимых обработать конвейеру

строка: «3: 1 1 -1 0 0 S = 4»

число 3 – это номер шага конвейера

1 1 -1 0 0 – это состояния каждого ИМ, всего их 5, как и количество ИМ. В данном случае ИМ1 и ИМ2 успешно поработали с деталью, ИМ3 при работе с деталью получил ошибку, ИМ4 и ИМ5 еще не были задействованы, они ожидают.

S = 4 – число четыре обозначает количество штатно работающих ИМ, то есть те, которые имеют положительные состояния: ожидание (0) и успешная работа с деталью (1).

Пример правильной работы программы, где пользователь пытался ввести значения отличные от тех, которые ожидает программа, а также с отказом Оператора от «починки ИМ». В следствие чего, конвейер был остановлен.

```
How many steps should the pipeline perform?
25
25
Enter the number_implementers (sections) in the pipeline(Conveyor):
15
15
Incorrect input, the number_implementers must be in the range: [1,10]
Enter the number_implementers (sections) in the pipeline(Conveyor):
3
3
Enter the number_details to process:
20
20
Incorrect input, the number_details must be in the range: [1,10]
Enter the number_details to process:
10
10
Number of implementers = 3
Number of details = 10
 0:  0  0  0  S = 3
 1:  1  0  0  S = 3
 2:  1  1  0  S = 3
 3:  1  1  1  S = 3
 4:  1  1  1  S = 3
 5:  1  1  1  S = 3
 6:  1  1  1  S = 3
 7:  1  1  1  S = 3
 8: -2  1  1  S = 2
There was an accident. Hardware error. In step 8
Continue automatic operation? (1 - Yes, 0 - No)
0
0
 Conveyor stop
work of the destructor

Process finished with exit code 0
```

Пример правильной работы программы, где пользователь использовал конструктор с параметрами. После получения ошибки от ИМ оператор решил осуществить «починку» ИМ. В следствие чего конвейер завершил обработку всех деталей.

```
How many steps should the pipeline perform?
20
20
Enter the number_implementers (sections) in the pipeline(Conveyor):
5
5
Enter the number_details to process:
10
10
Number of implementers = 5
Number of details = 10
 0:  0  0  0  0  0  S = 5
 1:  1  0  0  0  0  S = 5
 2:  1  1  0  0  0  S = 5
 3:  1  1 -1  0  0  S = 4
There was an accident. Hardware error. In step 3
Continue automatic operation? (1 - Yes, 0 - No)
1
1
Implementer as been fixed. An action has already been performed on the part.
 4:  1  1  1  1  0  S = 5
 5:  1  1  1  1  1  S = 5
 6:  1  1  1  1  1  S = 5
 7:  1  1  1  1  1  S = 5
 8:  1  1  1  1  1  S = 5
 9:  1  1  1  1  1  S = 5
10:  1  1  1  1  1  S = 5
11:  0  1  1  1  1  S = 5
12:  0  0  1  1  1  S = 5
13:  0  0  0  1  1  S = 5
14:  0  0  0  0  1  S = 5
15:  0  0  0  0  0  S = 5
16:  0  0  0  0  0  S = 5
17:  0  0  0  0  0  S = 5
18:  0  0  0  0  0  S = 5
19:  0  0  0  0  0  S = 5
 The pipeline has completed all the work
work of the destructor

Process finished with exit code 0
```