«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
- Implementer *ptr_Implementer = nullptr //дин массив для ИМ
- Detail *ptr_Detail = nullptr; //дин массив для Деталей

и имеет методы:

- конструктор с параметрами с вызовом исключений
- void static repair() – ремонтирует отдельный ИМ, с вызовом функции technological_operation
- uint16_t crash() const – вызывает оператора для определения действия починить ИМ или завершить работу конвейера
- operators_reply – функция для пользовательского ввода о принятии решения, ремонтировать ли ИМ или же остановить конвейер
- 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

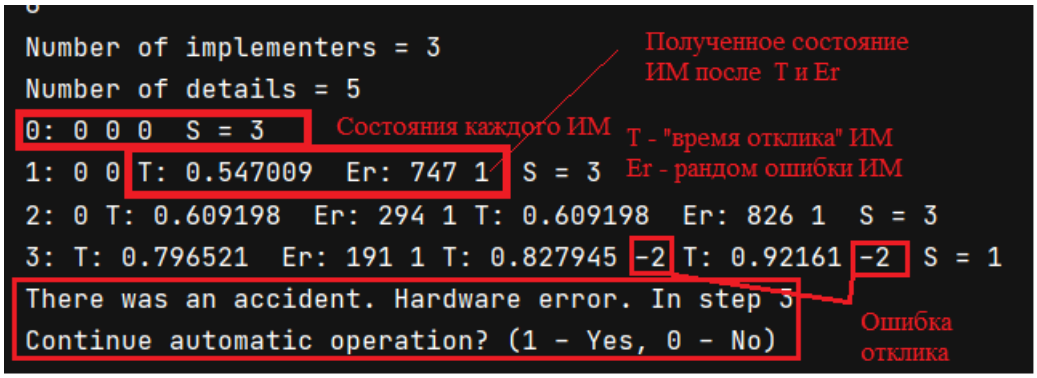
  state будет принимать значения из:

```
enum states{            //состояния для ИМ
   wait,                //0  ИМ ожидает деталь
   success,             //1  ИМ успешно отработал с деталью
   busy                 //3  ИМ будет использоваться на текущем шаге
   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
};
```

и имеет методы:

- конструктор по умолчанию
- int get_state() const – получение состояния ИМ
- void set_state() - установка нового значения состояния ИМ
- int technological_operation() – определяет с помощью вероятностей и рандомизации, каков будет результат работы ИМ над деталью

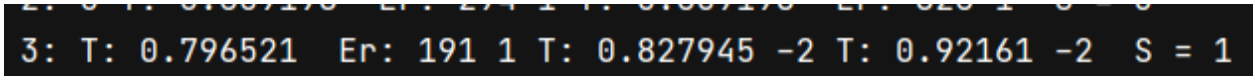
Разбор вывода состояния конвейера и его ИМ



Number of implementers = 3 – количество ИМ в конвейере

Number of details = 4 – количество деталей необходимых обработать конвейеру

строка:



число 3: – это номер шага конвейера

T – «время отклика ИМ» Er – рандомизация ошибки ИМ, 1 – результирующее состояние ИМ, после снова T Er и состояние -2, соот. Ошибке вида «слишком долгий отклик ИМ» S=1 – показывает, сколько ИМ справились успешно.

Пример правильной работы программы. 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?
10
10
Number of implementers = 3
Number of details = 5
0: 0 0 0  S = 3
1: 0 0 T: 0.547511  Er: 747 1  S = 3
2: 0 T: 0.60999  Er: 294 1 T: 0.60999  Er: 826 1  S = 3
3: T: 0.796663  Er: 191 1 T: 0.827779 -2 T: 0.921729 -2  S = 1
There was an accident. Hardware error. In step 3
Continue automatic operation? (1 - Yes, 0 - No)
1
1
Implementer [0] is state = -2 Repair..
T: 0.187324  Er: 43 1 New state is 1
Implementer as been fixed. An action has already been performed on the part.
Implementer [1] is state = -2 Repair..
T: 0.921658 -2 New state is -2
T: 0.0781094  Er: 91 1 Trying again. New state is 1
Implementer as been fixed. An action has already been performed on the part.
4: T: 0.0312278  Er: 68 1 T: 0.0937078  Er: 199 1 T: 0.749772  Er: 385 1  S = 3
5: T: 0.40613  Er: 524 1 T: 0.453077  Er: 625 1 T: 0.593846  Er: 130 1  S = 3
6: 0 T: 0.0619222  Er: 58 1 T: 0.983918 -2  S = 2
There was an accident. Hardware error. In step 6
Continue automatic operation? (1 - Yes, 0 - No)
1
1
Implementer [2] is state = -2 Repair..
T: 0.344242  Er: 766 1 New state is 1
Implementer as been fixed. An action has already been performed on the part.
7: 0 0 T: 0.0782275  Er: 818 1  S = 3
8: 0 0 0  S = 3
9: 00 0   S = 3
 The pipeline has completed all the work
~Implementer
~Implementer
~Implementer
~Conveyor

Process finished with exit code 0
```