**WEEK 12**

**KU ID: 100058537**

**NAME: Mariam Alzaabi**

|  |
| --- |
| Q1.  Please write your GitHub Repository link for your week 12 Rustlings submission: |
| ANSWER: |
| Q2.   1. Please **Find 1 (one)** specific part of the c2rust code that uses an **iterator** 2. and explain **why** that part uses an iterator **instead of a for loop**   -Important: Please also mention the **URL** where you found that iterator  -Please find **only in \*.rs** file(s) (not the \*.md, \*.html, \*.c, \*.h, etc. ) |
| ANSWER:  a.  pub fn get\_fields(&self) -> Vec<&CField> {  self.decls  .iter()  .filter\_map(|decl| match decl.kind {  CDeclKind::Field { ref field } => Some(field),  \_ => None,  })  .collect()  }  b. Explanation of why an iterator is used instead of a for loop:   1. Functional Style and Conciseness:   The iterator chain (.iter().filter\_map().collect()) expresses the intent more declaratively than an imperative for loop  It combines multiple operations (iteration, filtering, mapping, and collecting) in a single expression  A for loop equivalent would require more lines and explicit variable management:  // For loop equivalent would look like:  pub fn get\_fields(&self) -> Vec<&CField> {  let mut result = Vec::new();  for decl in &self.decls {  if let CDeclKind::Field { ref field } = decl.kind {  result.push(field);  }  }  result  }   1. Lazy Evaluation:    * Iterators process elements on-demand as collect() consumes them    * This is more efficient than a for loop that would process all elements immediately    * Particularly beneficial since self.decls could contain many declarations, but we only want fields 2. Type Safety and Clarity:    * The iterator's filter\_map clearly communicates that we're transforming CDecl into Option<&CField> and filtering out non-field declarations    * The return type Vec<&CField> is directly constructed via collect(), avoiding manual vector manipulation    * This reduces the chance of errors compared to pushing to a vector in a loop 3. Chainability:    * Using iterators allows easy extension of the operation chain if needed (e.g., adding .take(n) or .map(...))    * A for loop would require more structural changes to add such modifications   In this context, the iterator approach is preferred because it’s more idiomatic Rust, safer, and more maintainable. The C2Rust project often deals with complex AST transformations, making iterators' composability particularly valuable over traditional loops. |