Lab 2 documentation

hash_table.rs

Structs and Enums

1. **`Entry<Key, Value>` enum**:
   - Represents a single slot in the hash table, which can either be `Vacant` (empty) or `Occupied` (holding a key-value pair).
   - Methods:
     - `take`: Removes the value from an occupied entry and makes it vacant.
     - `replace`: Replaces the value in an occupied entry with a new one, returning the old value.
2. **`HashTable<Key, Value>` struct**:
   - The main structure that holds the hash table.
   - Key properties:
     - `entries`: A vector of `Entry` structs that represent the slots.
     - `occupied_size`: Tracks the number of occupied entries.
     - `vacant_size`: Tracks the number of vacant entries.
   - Supports generic key and value types, with the key required to implement the `Eq` (can be compared) and `Hash` (can be hashed) traits.

Methods

1. **`new`**:
   - Creates an empty hash table.
2. **`len`**:
   - Returns the number of occupied entries (i.e., the size of the hash table).
3. **`get_index<Q>(&self, key: &Q)`**:

- Hashes the key and returns the index in the `entries` vector. Uses a `DefaultHasher`.

4. **`iter_mut_starting_at(&mut self, index: usize)`**:
   - Returns a mutable iterator over the entries, starting from a specific index, which is helpful for handling collisions.
5. **`get_load_factor`**:
   - Computes the current load factor (`occupied_size / total_capacity`).
6. **`resize`**:
   - Dynamically adjusts the size of the hash table by doubling the capacity when the load factor exceeds the threshold (0.75). Rehashes all entries into a new vector of a larger size.
7. **`insert_without_resize`**:
   - Inserts a key-value pair without checking or performing a resize. It handles collisions via linear probing and returns the old value if the key already exists.
8. **`insert`**:
   - Inserts a key-value pair, resizing the table if necessary to maintain a good load factor.
9. **`contains_key<Q>(&self, key: &Q)`**:
   - Checks whether a given key exists in the table.
10. **`iter`**:
    - Returns an iterator over all occupied key-value pairs in the table.
11. **`get<Q>(&self, key: &Q)`**:
    - Retrieves the value associated with the given key (if it exists) using linear probing to resolve collisions.
12. **`get_mut<Q>(&mut self, key: &Q)`**:
    - Retrieves a mutable reference to the value associated with the given key.
13. **`remove<Q>(&mut self, key: &Q)`**:

- Removes the key-value pair corresponding to the provided key, returns the value if it was found, and marks the entry as vacant.

symbol_table.rs

Struct

- `SymbolTable`:
  - **Fields**:
    - `hash_table`: An instance of a hash table (`HashTable<String, i32>`) that maps tokens (strings) to unique integer positions (i32).
    - `last_index`: A counter that tracks the last assigned index for a symbol. It starts at `1` and increments each time a new symbol is added.

Methods

1. `new()`:
   - Initializes an empty `SymbolTable` with an empty hash table and sets `last_index` to `-1`.
2. `len()`:
   - Returns the number of symbols (i.e., key-value pairs) currently stored in the symbol table.
3. `insert_symbol(&mut self, token: String)`:
   - Inserts a new symbol (token) into the symbol table.
   - If the symbol already exists, it does nothing.
   - If the symbol is new, it increments `last_index` and associates the token with the new index.
4. `get_position_of_symbol(&self, token: String) -> Option<&i32>`:
   - Looks up the index associated with a given symbol (token).
   - Returns `Some(&i32)` if the symbol is found, or `None` if the symbol does not exist.

5. `get_symbol_at_position(&self, given_position: &i32) -> Option<&String>`:
   - Looks up the symbol that is associated with a given position.
   - Iterates through the entries in the hash table and returns `Some(&String)` if the position is found, or `None` if no symbol matches the given position.
6. `remove(&mut self, key: String)`:
   - Removes the symbol (token) from the symbol table, along with its associated position.
7. `contains_token(&self, token: String) -> bool`:
   - Checks if the given symbol (token) exists in the symbol table.
   - Returns `true` if the token is present, otherwise `false`.