RUST 프로그래밍

20.2.22
김지현

참고자료 :
https://github.com/TheAlgorithms/Rust

LinkedList 구현

-trait 선언

```
//출력을 위한 trait
use std::fmt::{self, Display, Formatter};
//nonnull을 쓰는 이유? 느리게 할당되는 유형을 초기화하는데 유용
use std::ptr::NonNull;
```

+ self

- 함수가 값을 소모해야 한다면 `self`를 쓰세요.
- 함수가 값에 대한 읽기 전용 참조만 필요하다면 `&self`를 쓰세요.
- 함수가 값을 소모하지 않으면서 값을 변경해야 한다면 `&mut self`를 쓰세요.

LinkedList 구현

```rust
//node 구조체
struct  Node<T>{
    val:T,
    next: Option<NonNull<Node<T>>>,
    prev: Option<NonNull<Node<T>>>,
}

impl <T> Node<T>{
    fn new(t:T) -> Node<T>{
        Node{
            val:t,
            //rust에는 null이 없음.
            prev:None,
            next:None,
        }
    }
}
```

-노드 구조체 선언

val, next, prev를 가지는 구조체 생성

-노드 메소드 선언

new 함수를 추가하여 빈 노드를 생성할 수 있도록 함

LinkedList 구현

```rust
pub struct LinkedList<T>{
    length: u32,
    start: Option<NonNull<Node<T>>>,
    end: Option<NonNull<Node<T>>>,
}

//여러 타입으로 LinkedList 생성 가능
impl<T> Default for LinkedList<T>{
    fn default() ->Self{
        Self::new()
    }
}
```

-Linkedlist 구조체

length, start, end 를 가지는 구조체 생성

-Linkedlist 메소드

default 함수 추가를 통해 여러 타입의 Linkedlist 생성할 수 있음

+default기능: 타입에 대한 기본값을 반환

LinkedList 구현

```
//초기화
impl<T> LinkedList<T>{
    pub fn new()->Self{
        Self{
            length:0,
            start:None,
            end:None,
        }
    }

    pub fn add(&mut self, obj: T){//뒤에 원소 추가
        //box로  힙 데이터를 참조 할 수 있음
        let mut node : Box<Node<T>> = Box::new( x: Node::new( t: obj));
        node.next = None;
        node.prev = self.end;
        //unsafe : 안전하지 않은 러스트로 전환
        //new_unchecked : unsafe로서 사용자가 그 데이터의 재배치나 무효화를 하지 않을 책임을 짐
        let node_ptr = Some(unsafe{NonNull::new_unchecked( ptr: Box::into_raw( b: node))});
        match self.end {
            //마지막 원소가 없으면, 시작위치에 넣기
            None =>self.start = node_ptr,
            //마지막 원소가 있으면, 마지막 원소의 다음에 넣기
            Some(end_ptr : NonNull<Node<T>> ) => unsafe{(*end_ptr.as_ptr()).next = node_ptr},
        }
        //마지막 원소에 업데이트
        self.end = node_ptr;
        //길이 업데이트
        self.length +=1;
    }
```

+unsafe 블록 내에서 가능한일

- 로우 포인터 (raw pointer) 를 역참조하기
- 안전하지 않은 함수 혹은 메소드 호출하기
- 가변 정적 변수 (mutable static variable) 의 접근 혹은 수정하기
- 안전하지 않은 트레잇 구현하기

-Linkedlist 메소드

- new 함수: Linkedlist를 반환하는 빈 생성자 생성 가능
- add 함수 : 새로운 노드를 heap에 선언하고 새로운 원소를 맨 뒤에 추가함

LinkedList 구현

```
    pub fn get(&mut self, index: i32) -> Option<&T>{
        self.get_ith_node( node: self.start, index)
    }

    fn get_ith_node(&mut self, node:Option<NonNull<Node<T>>>, index: i32) ->Option<&T>{
        match node{
            None =>None,
            Some(next_ptr : NonNull<Node<T>> ) => match index{
                //인덱스가 0이면 첫번째 원소 반환
                0=> Some(unsafe{ &(*next_ptr.as_ptr()).val}),
                //인덱스가 1이상이면 인덱스 만큼 get_ith_node호출
                _=> self.get_ith_node( node: unsafe { (*next_ptr.as_ptr()).next }, index: index - 1),
            },
        }
    }
}
```

- get함수: index 번째 원소를 가져옴 (public)
- get_ith_node 함수: match를 이용해 index 번째 원소를 반환 (private)

LinkedList 구현

```
//링크드리스트 출력을 위한 메소드
impl<T> Display for LinkedList<T>
    where
        T:Display,
{
    fn fmt(&self, f:&mut Formatter) ->fmt::Result{
        match self.start{
            //start의 노드 출력
            Some(node : NonNull<Node<T>> ) => write!(f,"{}", unsafe{node.as_ref()}),
            None=>Ok(()),
        }
    }
}
//노드 출력을 위한 메소드
impl<T> Display for Node<T>
    where
        T:Display,
{
    fn fmt(&self, f: &mut Formatter) -> fmt::Result {
        match self.next {
            //현재 노드의 원소와 next가 가리키는 노드 출력
            Some(node : NonNull<Node<T>> ) => write!(f,"{}, {}",self.val, unsafe{node.as_ref()}),
            //가리키는 다음 노드가 없다면 현재 노드의 원소만 반환
            None=>write!(f,"{}",self.val),
        }
    }
}
```

-출력을 위한 메소드(Display)

- Linkedlist출력: start에 있는 노드정보 출력
- Node출력: 노드의 원소 정보 출력

LinkedList 구현

```rust
#[cfg(test)]
mod tests{
    use super::LinkedList;

    #[test]
    fn create_numeric_list(){
        //링크드리스트 생성
        let mut list : LinkedList<i32> = LinkedList::<i32>::new();
        //링크드리스트 원소 추가
        list.add( obj: 1);
        list.add( obj: 2);
        list.add( obj: 3);
        //링크드리스트 출력
        println!("Linked List is {}",list);
        assert_eq!(3, list.length);
    }

    #[test]
    fn get_by_index_in_numeric_list() {
        let mut list : LinkedList<i32> = LinkedList::<i32>::new();
        list.add( obj: 1);
        list.add( obj: 2);
        println!("Linked List is {}", list);
        let retrived_item : Option<&i32> = list.get( index: 0);
        //원소가 잘 가져와졌는지 검사
        assert!(retrived_item.is_some());
        //가져온 원소가 1이 맞는지 검사
        assert_eq!(1 as i32, *retrived_item.unwrap());
    }

}
```

-test 모듈

- add 기능 테스트: 원소 추가후 내용 출력, 길이 검사
- get 기능 테스트: 원소 추가후 0번째 원소 가져와 맞는지 검사

```
    Finished test [unoptimized + debuginfo] target(s) in 0.09s
     Running target\debug\deps\webserver-199e2d5d8a033416.exe
Linked List is 1, 2, 3
```

```
    Finished test [unoptimized + debuginfo] target(s) in 0.09s
     Running target\debug\deps\webserver-199e2d5d8a033416.exe
Linked List is 1, 2
```

```
    Finished test [unoptimized + debuginfo] target(s) in 0.09s
     Running target\debug\deps\webserver-199e2d5d8a033416.exe
Linked List is 1, 2


Left:  2
Right: 1
<Click to see difference>
```