**Title:** *Traits*

**Name(s):**

*Chanasorn Howattanakulphong 65011277*

**Aim:**

*Experiment and get used to using traits in rust by imitating an organization with different people that contain same traits.*

Methods**:**

*First I think of what structs I’m going to make that can use same traits. I chose teacher, doctor, and admiral. Our employee structs will have some of the same properties, such as name, age, and location. We already have the GPS struct from prior labs, so we are using that as our location for these new structs. The job’s characteristic must contain at least one list, so for teacher it is the list of students in his class right now. Same goes for doctor and admiral, they have their ships and patients.*

*The trait will include functions that prints out each detail of the struct. Such as getName(), getAge(), getLocation(), and getResponsibility().(getResponsibility is for calling the list in the structs). Then I just made a function to read the input, then save all the information inside the struct and displays them.*

**Results:**

*Text

Description automatically generated*

*Text

Description automatically generated*

*Text

Description automatically generated*

**Discussion:**

*The results are as I expected them to be, since it’s just inputting into structs and then the program will save them and display the attributes of each of them.*

**Conclusion:**

*The program works according to the aim of this lab, which is to share traits between different structs and call them And the output also displays all the attributes required including the list in each structs.*

**Acknowledgments:**

*I acknowledge the help and guidance from my teachers, friends, and Tas.*

**Appendix:**

use std::fmt;

use std::fmt::Formatter;

use std::io;

use std::io::\*;

fn main(){

vals = Vec::new();

  let mut count = 0;

  let mut structt = String::new();

  let mut agesum = 0;

  let mut person = 0;

  loop {

      //eprintln!("check loop");

      let mut text = read\_text\_line();

      if text.len() < 3 { break; }

      else {

          let n = text.split(": ");

          let str\_vec = n.collect::<Vec<&str>>();

          if str\_vec[0].to\_string() == "job".to\_string(){

              if count != 0{

                  printt(structt, vals[1..vals.len()].to\_vec());

                  vals.clear();

              }

              structt = str\_vec[1].to\_string();

              person += 1;

          }

          if str\_vec[0].to\_string() == "age".to\_string(){

              agesum += str\_vec[1].parse::<u32>().unwrap();

          }

          vals.push(str\_vec[1].to\_string());

          //eprintln!("Vals => {:?}", vals[1..vals.len()].to\_vec());

      }

      count += 1;

  }

  printt(structt, vals[1..vals.len()].to\_vec());

  println!("Average age: {}", agesum/person);

}

#[derive(Debug, Copy, Clone)]

struct GPS {

  lati: f64,

  long: f64

}

fn show\_position(gps : GPS) {

  const Degree: &str = "\u{00b0}";

  let position = gps;

  let mut lati\_direction = "N";

  let mut long\_direction = "E";

  if position.lati < 0.0 {

      lati\_direction = "S";

  }

  if position.long < 0.0 {

      long\_direction = "W";

  }

  print!{"{}{} {}, {}{} {}", position.lati, Degree, lati\_direction, position.long, Degree, long\_direction}

}

#[derive(Debug, Clone)]

struct teacher{

  name : String,

  age : u32,

  location : GPS,

  students : Vec::<String>,

}

#[derive(Debug, Clone)]

struct admiral{

  name : String,

  age : u32,

  location : GPS,

  ships : Vec::<String>,

}

#[derive(Debug, Clone)]

struct doctor{

  name : String,

  age : u32,

  location : GPS,

  patients : Vec::<String>,

}

trait employees {

  fn getName(&self);

  fn getAge(&self);

  fn getLocation(&self);

  fn getResponsibility(&self);

}

impl employees for teacher{

  fn getName(&self){

    println!("Teacher : {}", self.name);

  }

  fn getAge(&self){

    println!("Age : {}", self.age);

  }

  fn getLocation(&self){

    println!("Location : {}, {}", self.location.lati, self.location.long);

  }

  fn getResponsibility(&self){

      println!("Current Students : {:?}", self.students)

  }

}

impl employees for admiral{

  fn getName(&self){

    println!("Admiral : {}", self.name);

  }

  fn getAge(&self){

    println!("Age : {}", self.age);

  }

  fn getLocation(&self){

    println!("Location : {}, {}", self.location.lati, self.location.long);

  }

  fn getResponsibility(&self){

      println!("Current Ships : {:?}", self.ships)

  }

}

impl employees for doctor{

  fn getName(&self){

    println!("Doctor : {}", self.name);

  }

  fn getAge(&self){

    println!("Age : {}", self.age);

  }

  fn getLocation(&self){

    println!("Location : {}, {}", self.location.lati, self.location.long);

  }

  fn getResponsibility(&self){

    println!("Current Patients : {:?}", self.patients)

  }

}

fn read\_text\_line() -> String {

  let mut buffer = String::new();

  let \_result = io::stdin().read\_line(&mut buffer);

  buffer = buffer.trim().to\_string();

  // eprintln!("Buffer read ({}) [{}]", buffer.len(), buffer );

  buffer

}

fn inp(structt: String, val: Vec<String>) -> (teacher, admiral, doctor){

  let g = GPS{

    lati: 0.0,

    long: 0.0,

  };

  let mut t1 = teacher{

      name: String::from(""),

      age: 0,

      location: g,

      students: Vec::new(),

  };

  let mut a1 = admiral{

      name: String::from(""),

      age: 0,

      location: g,

      ships: Vec::new(),

  };

  let mut d1 = doctor{

      name: String::from(""),

      age: 0,

      location: g,

      patients: Vec::new(),

  };

  if structt == "Teacher".to\_string(){

      let lat\_arr = val[2].split(", ").collect::<Vec<&str>>();

      t1 = teacher{

        name: val[0].clone(),

        age: val[1].parse::<u32>().unwrap(),

        location: GPS{

          lati: lat\_arr[0].parse::<f64>().unwrap(),

          long: lat\_arr[1].parse::<f64>().unwrap(),

        },

        students: val[3].split(",").map(|s| s.to\_string()).collect(),

      };

  }

  else if structt == "Admiral".to\_string(){

      let lat\_arr = val[2].split(", ").collect::<Vec<&str>>();

      a1 = admiral{

        name: val[0].clone(),

        age: val[1].parse::<u32>().unwrap(),

        location: GPS{

          lati: lat\_arr[0].parse::<f64>().unwrap(),

          long: lat\_arr[1].parse::<f64>().unwrap(),

        },

        ships: val[3].split(",").map(|s| s.to\_string()).collect(),

      };

  }

  else if structt == "Doctor".to\_string(){

      let lat\_arr = val[2].split(", ").collect::<Vec<&str>>();

      d1 = doctor{

        name: val[0].clone(),

        age: val[1].parse::<u32>().unwrap(),

        location: GPS{

          lati: lat\_arr[0].parse::<f64>().unwrap(),

          long: lat\_arr[1].parse::<f64>().unwrap(),

        },

        patients: val[3].split(",").map(|s| s.to\_string()).collect(),

      };

  }

  return (t1, a1, d1);

}

fn printt(structt: String, val: Vec<String>){

  let (t1, a1, d1) = inp(structt, val.clone());

  print!("\n\n");

  if t1.name != "".to\_string(){

      t1.getName();

      t1.getAge();

      t1.getLocation();

      t1.getResponsibility();

  }

  else if a1.name != "".to\_string(){

      a1.getName();

      a1.getAge();

      a1.getLocation();

      a1.getResponsibility();

  }

  else if d1.name != "".to\_string(){

      d1.getName();

      d1.getAge();

      d1.getLocation();

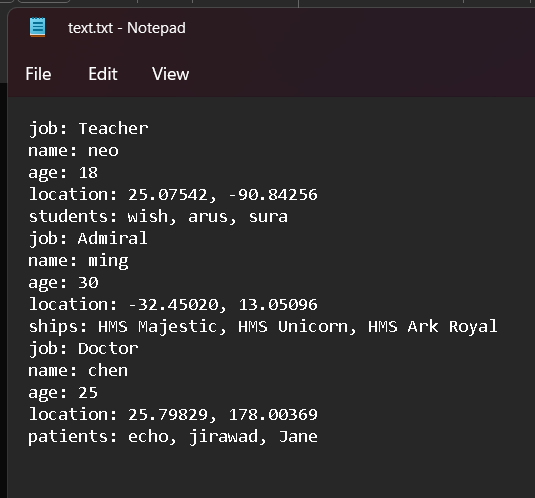
      d1.getResponsibility();

  }

  print!("\n\n");

}

*Inputs:*

**