CMPT 383: Vitamin #11

Anders Miltner
miltner@cs.sfu.ca

Due Nov 30

Introduction

This Vitamin is to help you practice using unsafe Rust. The test suite is provided in `src/lib.rs`. You should fill out the function definitions in `src/functions.rs`.

This submission will be partially autograded. There are some portions of the assignment that are ungraded, and some that will be graded. We provide a (partial) test suite for partial validation. You can run these tests by opening a terminal in the v11 directory, and running `cargo test`.

We have included all relevant imports. If you import additional functions, you may get a zero on the assignment. You will be implementing a variable-sized array in unsafe Rust.

1 Components Used

In the implementation of the array, you will need a number of built-in functions. This section details those functions. Clicking on the italicized text brings you to the Rust documentation on the function.
alloc. Alloc takes a Layout, describing the shape of required memory, as an input, and will heap-allocate memory in that layout.
dealloc. Dealloc takes a pointer, and a Layout describing the shape of required memory, as an input, and will de-allocate the memory at that pointer that conforms to that Layout.
add. Add takes a pointer, and an offset, and will return the pointer associated with that offset (in C terms, it performs basic pointer adding).
write. Write takes a pointer, and a value, and will write the value to the location in memory pointed to.

2 `new`

To write the `new` function, you must create an Array struct. The Array struct consists of two things, a mutable pointer and a length. The length is provided to the `new` function. The pointer should be created using the `alloc` function. The `Layout::array::<T>(size)` function will return a layout corresponding to a sufficiently large chunk of memory for an array of `T` of length `size`.

3 `element_at`

To write the `element_at` function, you must reference the raw pointer of the Array. You first do pointer arithmetic using the `add` function to get the correct pointer. After getting the correct pointer `p` for the index, you can extract the value with `&*p`. You should manually perform a bounds check, and panic if the index is out of bounds.

4 `set`

To write the `set` function, you must reference the raw pointer of the Array. You first do pointer arithmetic using the `add` function to get the correct pointer. After getting the correct pointer `p` for the index, you can set that to a new value with the `write` function. You should manually perform a bounds check, and panic if the index is out of bounds.

5 `drop`

To write the `drop` function, you must deallocate the raw pointer, with the same Layout used to allocate.