2. Vector CDTS

- Recall the constraints on allocating arrays on the stack, and the need to explicitly pass the array size
- Let’s transform a dynamically allocated array along with its maximum size and actual size into a data structure

2.1. Vector CDT Interface

```c
typedef struct
{
    // implementation specific
}
vector_int_t;
```

- `void vector_int_construct ( vector_int_t* this );`
  Construct the vector initializing all fields in this `vector_int_t`.

- `void vector_int_destruct ( vector_int_t* this );`
  Destruct the vector by freeing any dynamically allocated memory used by this `vector_int_t`.

- `void vector_int_push_front ( vector_int_t* this, int v );`
  Push a new value (v) at the front of this `vector_int_t`.

- `void vector_int_reverse ( vector_int_t* this );`
  Reverse all values in this `vector_int_t`.  

Example of using vector interface

```c
int main( void )
{
    vector_int_t vector;
    vector_int_construct ( &vector );
    vector_int_push_front( &vector, 12 );
    vector_int_push_front( &vector, 11 );
    vector_int_push_front( &vector, 10 );
    vector_int_reverse ( &vector );
    vector_int_destruct ( &vector );
    return 0;
}
```

2.2. Vector CDT Implementation

```c
typedef struct
{
    int* data;
    size_t maxsize;
    size_t size;
}
vector_int_t;
```

- To simplify our discussion in lecture, we will allocate a small array in the constructor, and then we assume we only push back a limited number of values
Approach for implementing functions

1. Draw figure to explore high-level approach
2. Develop pseudo-code to capture high-level approach
3. Translate the pseudo-code to actual C code

Pseudo-code for `vector_int_construct`

```
1 void vector_construct( vector_int_t* this )
2     allocate new array with eight elements
3     set vector’s data to point to new array
4     set vector’s maxsize to eight
5     set vector’s size to zero
```

Pseudo-code for `vector_int_push_front`

Initial state of vector

After push front of value 9
After push front of value 8

Implement moving down all of the elements

```c
void vector_push_front( vector_int_t* this, int v )
    set prev value to v
    for i in 0 to vector’s size (inclusive)
        set temp value to vector’s data[i]
        set vector’s data[i] to prev value
        set prev value to temp value
    set vector’s size to size + 1
```

**Pseudo-code for vector_int_destruct**

```c
void vector_destruct( vector_int_t* this )
    free vector’s data
```
2. Vector CDTs

2.2. Vector CDT Implementation

// Construct vector
void vector_int_construct(
    vector_int_t* this)
{
    this->data = malloc( 8 * sizeof(int) );
    this->maxsize = 8;
    this->size = 0;
}

// Push value on front of vector
void vector_int_push_front(
    vector_int_t* this, int v)
{
    int prev_value = v;
    for (size_t i=0; i<=this->size; i++) {
        int temp_value = this->data[i];
        this->data[i] = prev_value;
        prev_value = temp_value;
    }
    this->size += 1;
}

// Destruct vector
void vector_int_destruct(
    vector_int_t* this)
{
    free(this->data);
}

// Main function
int main( void )
{
    vector_int_t vector;
    vector_int_construct( &vector);
    vector_int_push_front( &vector, 12 );
    vector_int_push_front( &vector, 11 );
    vector_int_push_front( &vector, 10 );
    vector_int_destruct( &vector);
    return 0;
}

https://repl.it/@cbatten/
ece2400-T07-ex2
2. Vector CDTs

2.2. Vector CDT Implementation

Interface vs. Implementation

- Implementation details are exposed in `vector_int_t`
- A user can freely manipulate fields in `vector_int_t`
- C does not provide any mechanism to enforce encapsulation

Develop an algorithm for `vector_int_reverse`
3. Sequence ADTs

- Imagine putting together a music playlist
- We can insert songs into any position in the playlist
- We can remove songs from any position in the playlist
- We can access/change (get/set) songs anywhere in the playlist
- We can iterate through the playlist to play the music

Sequence ADT interface

- Pseudocode for working with a sequence ADT

```plaintext
1  insert 2 at beginning of sequence
2  insert 4 at end of sequence
3  insert 6 at end of sequence
4  insert 3 at beginning of sequence
5  set iterator to beginning of sequence
6  while iterator is not equal to end of sequence
    7  get value at iterator
    8  set iterator to next iterator
```
3. Sequence ADTs

- C-based interface for sequence ADT

```c
typedef struct
{
    // opaque
}
seq_t;

typedef /* opaque */     itr_t;
typedef /* user defined */ item_t;

void seq_construct ( seq_t* seq );
void seq_destruct ( seq_t* seq );
void seq_push_front( seq_t* seq, item_t v );
void seq_reverse ( seq_t* seq );
size_t seq_size ();
item_t seq_at ( size_t idx );
itr_t seq_begin ( seq_t* seq );
itr_t seq_end ( seq_t* seq );
itr_t seq_next ( seq_t* seq, itr_t itr );
item_t seq_get ( seq_t* seq, itr_t itr );
void seq_set ( seq_t* seq, itr_t itr, item_t v );
void seq_insert ( seq_t* seq, itr_t itr, item_t v );
void seq_remove ( seq_t* seq, itr_t itr );

int main( void )
{
    seq_t seq;
    seq_construct ( &seq );
    seq_push_front( &seq, 2 );
    seq_push_front( &seq, 4 );

    itr_t itr = seq_begin( &seq );
    while ( itr != seq_end( &seq ) ) {
        int value = seq_get( &seq, itr );
        itr = seq_next( &seq, itr );
    }

    seq_destruct( &seq );
    return 0;
}
```
3. Sequence ADTs

Sequence ADT implementations

• List implementation
  – dynamically allocated nodes and pointers
  – itr_t is a pointer to a node
  – seq_begin returns the head pointer
  – seq_end returns the NULL pointer
  – seq_next returns itr->next_p

• Vector implementation
  – dynamically allocated array (with resizing)
  – itr_t is an index
  – seq_begin returns 0
  – seq_end returns size
  – seq_next returns itr++

• List vs. Vector
  – Performance and storage for push_front?
  – Performance and storage for reverse?
  – Performance and storage for insert?
  – Performance and storage for remove?