Vector

← — — — alluding to what std::vector<T> in c++ will be.

Things we want:
- append ←
- length tracking
- resizing ←
- join two list easily ←
- equality
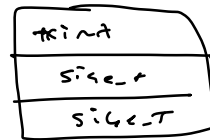
types!!!
*arr = arr[]
int *arr;
≤int arr[];

typedef struct ε
    int array [5];
    size len;t
...

3 Readl Example_T;

typedef struct ε   ← what alternative is ??
    int *array; —
    size_t length; =   — — — → size_t
    size_t capacity;

*int
size_t
size_t

→ integral type
→ ≥ 0 type

sizeof (vectorInt I)
= sizeof (int *)
+ sizeo (*size_t)
+ sizeof (size_t)

3 VectorInt_T ;
      typename

head
pad
len
cap

newVec

// malloc something
vectorInt_T* createVectorInt (size_t initalCapacity) ε

Imperative Style.

    VectorInT_T *newVec = (VectorInt_T *)malloc (sizeof(vectorInt_T));

→   newVec→array = (int *)malloc(initalCapacity * sizeof (int));
    newVec→capacity = inital Capacity;
    newVec→length = 0;

    return newVec;

3
void   free VectorInt (vectorInt_T *vec) ε

    free (vec→array);

    free (vec);

3

Creating with
Pointers

VectorInt_T   createVectorInt^stack (size_t initalCapacity) {
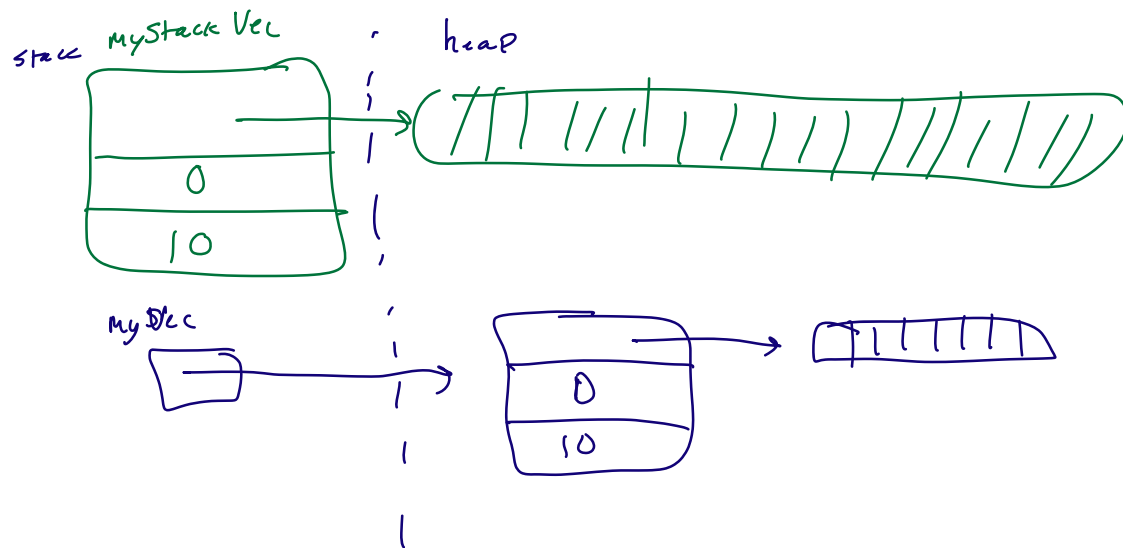
    VectorInt_T newVec;

    newVec.array = (int *)malloc(initalCapacity * sizeof (int));
    newVec.capacity = inital Capacity;

    newVec.length = 0;
    return NewVec;
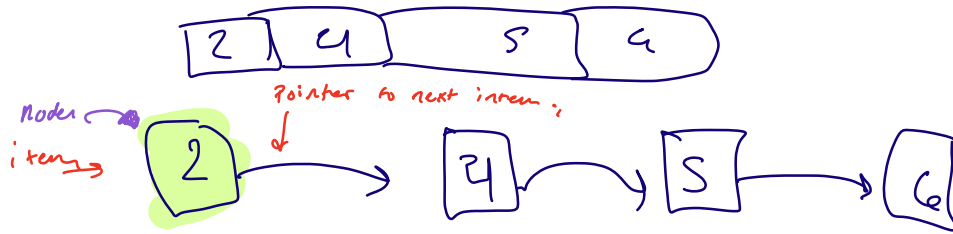
Functional Style

3

```
int main() {
    VectorInt_T myStackVec = createVecIntStack(10);
    VectorInt_T *myVec = createVectorInt(10);
    free(myStackVec.array);
    myStackVec.array = malloc(25 · sizeof int );
```

stack myStackVec   heap

myVec

What is a Linked List?

Lists are resizable, have no fixed size.

| 2 | 4 | 5 | 4 |

Pointer to next item;

Node →
item →

2 → 4 → 5 → 6

Linked → a type linked data structure

Ways of Writing Struct

typedef struct {
    // stuff

    3 (name_T;

int a = a + 3;

---

struct   type

typedef struct list_node list;
struct list_node {
    int item;
    list * next; ←
3;

| int item |
| pointer list |

---

typedef elem int;
typedef struct list_node list;
struct list_node {
    elem item;
    list* next;
3;

| 7 | . | → | 6 | |
| 9 | |
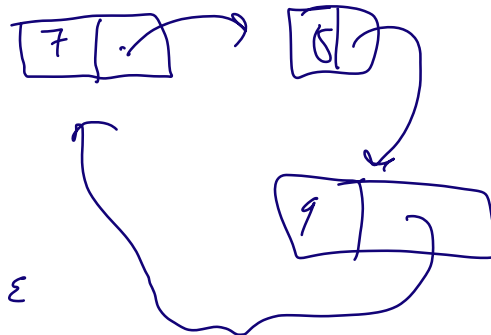
```
bool is_acyclic (list * start, list* end) {
    for (list * p = start; p != end; p = p→next) {
      → if (p == NULL) return True;
        for (list * r = start; r != p; r = r→next) {
            if ( p→next == r    return False;
3
3   return True;
3
```

```
bool    is_acyclic ( list * start,  list * end ) {
    if ( start == NULL)  return true;
    list *  h   =   start → next;
    list *  t   =   start;
    while ( h != t ) {
        if ( h == NULL || h → next == NULL || h → next
             return true;                        → next
                                                 == NULL)
        t = t → next;
        h = h → next → next;
    }
    return false;
}
```