

# Week 12 Lecture 2

## Dynamic Arrays

# Dynamic Arrays

# Arrays that grow as needed

- Dynamic arrays are arrays that get bigger as you need more space.
  - They are not part of C, but are relatively easy to implement.

# Interface

- `dynamic_array.h`
- `dynamic_array`
  - Size
  - Values
- Methods

```
#define TRUE 1
#define FALSE 0

typedef struct da {
    int *values;
    int size;
} dynamic_array;

dynamic_array new_dynamic_array();
int get(dynamic_array da, int index);
void set(dynamic_array da, int index, int value);
int expand(dynamic_array *da);
void print_dynamic_array(dynamic_array da);
```

Create new array: `new_dynamic_array()`;

- Access array: `get(dynamic_array d, int index)`;
- Add to array: `set(dynamic_array d, int index)`
- Expand array: `expand(dynamic_array d)`;

# Calls to Interface

- New
  - Size == 1
- Expand
  - Size == 2
- Set
- Expand again
  - Size == 4

```
#include <stdio.h>
#include "dynamic_array.h"

int main()
{
    dynamic_array da;

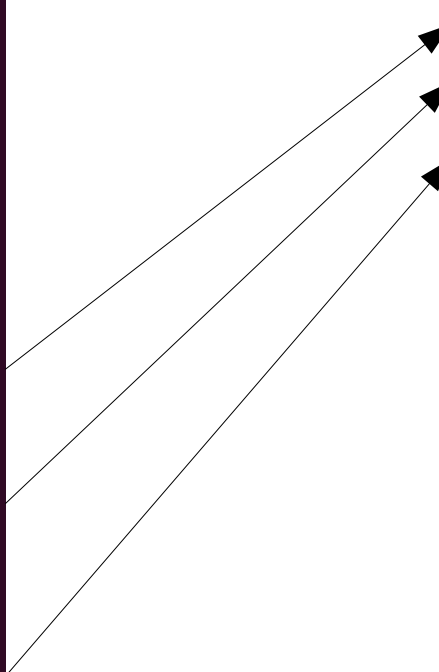
    da = new_dynamic_array();
    set(da, 0, 5);
    printf("%d\n", get(da,0));
    expand(&da);
    set(da, 1, 10);
    print_dynamic_array(da);
    expand(&da);
    set(da, 2, 12);
    set(da, 3, 13);
    print_dynamic_array(da);
    return 0;
}
```

# Results

```
#include <stdio.h>
#include "dynamic_array.h"

int main()
{
    dynamic_array da;

    da = new_dynamic_array();
    set(da, 0, 5);
    printf("%d\n", get(da,0));
    expand(&da);
    set(da, 1, 10);
    print_dynamic_array(da);
    expand(&da);
    set(da, 2, 12);
    set(da, 3, 13);
    print_dynamic_array(da);
    return 0;
}
```



```
5
{5, 10}
{5, 10, 12, 13}
```

# Create new array

- Pointer to new array
- New array created to put in pointer
- Size initialized to 1
- Array initialized to one int
- Return the object
  - Return \*da
  - Not the address da

```
dynamic_array new_dynamic_array()
{
    dynamic_array *da;

    da = malloc(sizeof(dynamic_array));
    da->size = 1;
    da->values = malloc(sizeof(int));
    return *da;
}
```

# New syntax ( $x \rightarrow y$ )

- Expression  $x \rightarrow y$  is the same as  $(*x).y$ .
  - If  $x$  is a pointer to a structure  $x \rightarrow y$  is the  $y$  field of the structure pointed to.
  - For example:

```
dynamic_array *da;  
da->size == (*da).size;
```



# Get

- Check for error
- Return the value from array

```
int get(dynamic_array da, int index)
{
    if (da.size < index) {
        printf("Get error: index %d bigger\n", index);
        return -1;
    }
    return da.values[index];
}
```

# Set

- Check for error
- Set array index to value provided

```
void set(dynamic_array da, int index, int value)
{
    if (da.size < index) {
        printf("Get error: index %d bigger than array size\n", index);
    }
    da.values[index] = value;
}
```

# Expand

- Pass in pointer to array
  - We need to manipulate the calling function's array.
- Double size
- Request new memory
- Copy old contents into new memory
- Free old memory
- Set array to new size and memory

```
int expand(dynamic_array *da)
{
    int new_size = da->size * 2;
    int *new_values;

    new_values = malloc(new_size);
    for (int i = 0; i < da->size; i++) {
        new_values[i] = da->values[i];
    }

    free(da->values);
    da->values = new_values;
    da->size = new_size;
    return TRUE;
}
```

# Print\_Dynamic\_Array

- Print each value from the beginning to the size.
  - Will print uninitialized values

```
void print_dynamic_array(dynamic_array da)
{
    printf("{");
    for (int i = 0; i < da.size; i++) {
        printf("%d%s", da.values[i], (i == da.size-1 ? "}\n" : ", "));
    }
}
```