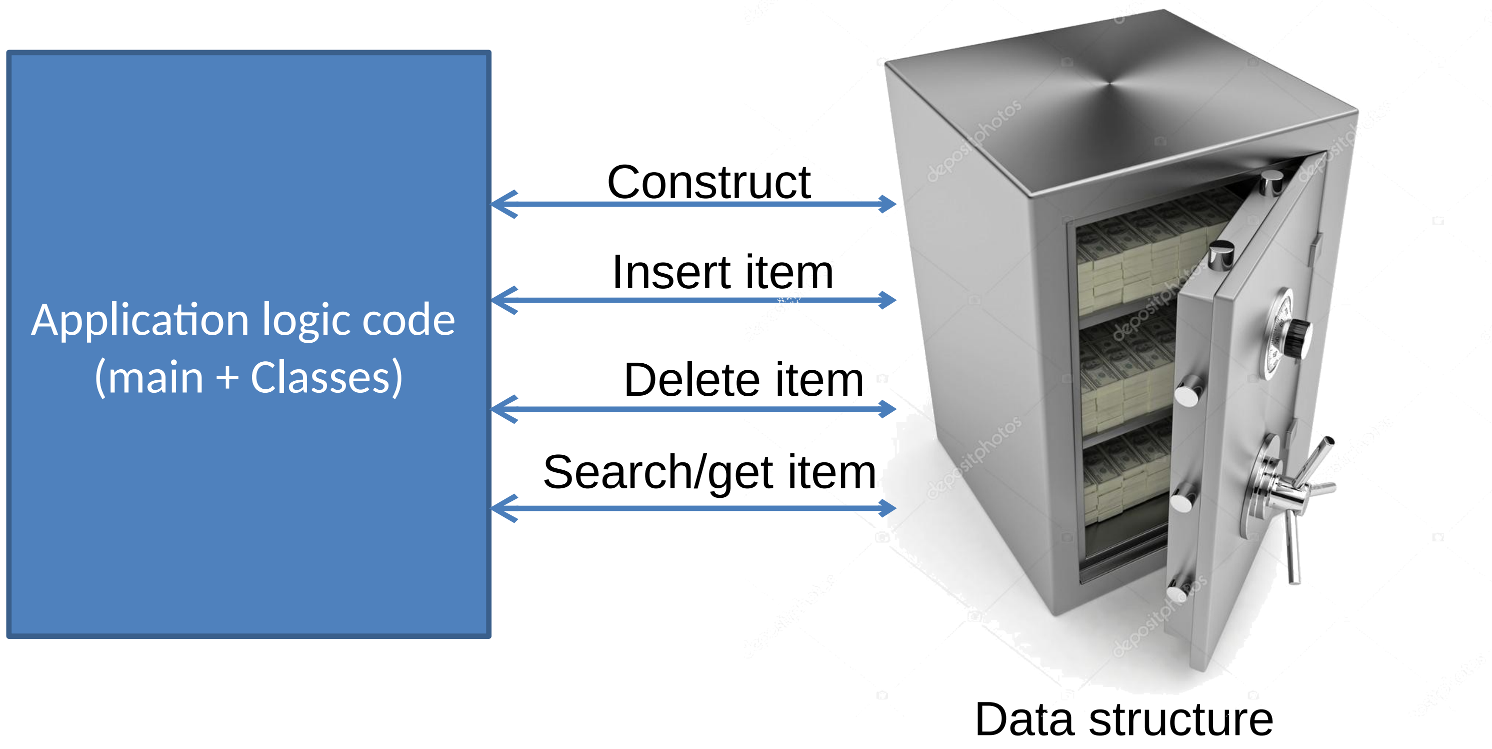
# CPSC 131
# Data Structures Concepts

Dr. Anand Panangadan
apanangadan@fullerton.edu

# Data structure



# Limitations of Arrays

//array of 10 ints: m[0], …, m[9]

int myArray[10];

- Size has to be fixed at compile time
- No error checking: Does not check that the index is valid: myArray[10]  segmentation fault
- 2 built-in arrays cannot be copied with =
  - int a[3] = {20, 40, 50};
  - int b[] = a; // This does NOT work

# Vectors

- Abstract use of arrays
  - Set length at runtime
  - Error checking of index bounds
  - Use dynamic memory allocation automatically
    - Implement within constructor/destructor
  - Number of elements ("size") is part of the data structure
    - size(), push_back(), pop_back()
  - Copying of vectors

# Arrays vs Vectors

```
class Person {
public:
  string name_;
  int age_;
  Person (string name, int age): name_(name), age_(age) { }
};
```

| | Arrays | Vectors |
|---|---|---|
| Create | `Person myfamily[100];` | `std::vector<Person> myfamily;` |
| Get/set element at index I | `cout << myfamily[i];`<br>`myfamily[i] = f;` | `cout << myfamily[i];`<br>`myfamily[i] = f;` |
| Max size | Must be specified when creating | `myfamily.resize(10);` |
| Add to end | must know last index | `myfamily.push_back(Family(“Anand”, 40);` |
| Number of elements | must use a separate “size” variable | `myfamily.size()` |

## Two C++ implementations

- std::vector
  - Part of the C++ Standard Library
  - #include <vector>
  - *Used* extensively in the real world
- CPSC 131 implementation
  - start with FixedVector
  - Improve to ExtendableVector
  - Meant to understand *how* a vector is implemented

## Fixed Length Vector

- Code shown in class that implements the FixedLengthVector class is posted on course GitHub page:

  https://github.com/CSUF-CPSC-131-Fall2019/Data-Structures-Code

# A Programming Problem

- Given a file of top 100 songs on Spotify
  - Each column represents:
  - name,artists,danceability,energy,key,loudness,mode,speechiness,acousticness,instrumentalness,liveness,valence,tempo,duration_ms,time_signature,rank
- Write a program to show songs
  - With danceability greater than 0.5
  - With danceability greater than average

## Hints

- First create a Song class
- Then use this Song class to instantiate the template class `std::vector<Song>`

## Vectors

# Limitations of Arrays

//array of 10 ints: m[0], …, m[9]

int myArray[10];

- Size has to be fixed at compile time
- No error checking: Does not check that the index is valid: myArray[10]  segmentation fault
- 2 built-in arrays cannot be copied with =
  - int a[3] = {20, 40, 50};
  - int b[] = a; // This does NOT work

## Vectors

- Abstract use of arrays
  - Set length at runtime
  - Error checking of index bounds
  - Use dynamic memory allocation automatically
    - Implement within constructor/destructor
  - Number of elements ("size") is part of the data structure
    - size(), push_back(), pop_back()
  - Copying of vectors

# Arrays vs Vectors

```
class Person {
public:
  string name_;
  int age_;
  Person (string name, int age): name_(name), age_(age) { }
};
```

| | Arrays | Vectors |
|---|---|---|
| Create | `Person myfamily[100];` | `std::vector<Person> myfamily;` |
| Get/set element at index I | `cout << myfamily[i];` `myfamily[i] = f;` | `cout << myfamily[i];` `myfamily[i] = f;` |
| Max size | Must be specified when creating | `myfamily.resize(10);` |
| Add to end | must know last index | `myfamily.push_back(Person("Tom", 20);` |
| Number of elements | must use a separate "size" variable | `myfamily.size()` |

# References

- CSUF CPSC 131 Slides, Dr. Anand Panangadan

# C++ Standard Library
# Vector

- Vectors are arrays of elements, similar to:
  - int x[10];
  - myClass c[5];
- Vectors are template classes; elements can be:
  - Built-in types like int.
  - Structs and classes
- Elements are accessed with the subscript operator []
- Vectors can change size to fit—they can grow and shrink as the program runs.

## Preparing to use the vector class

- #include <vector>
- using namespace std; *or*
- using std::vector;

## Defining vectors

- vector<type> name;
- vector<int> vector1;
- vector<myClass> vector2;

## Setting a vector's size

- vector<int>      vector1;     **// Size is 0**
- vector<int>      vector2(10);   **//   Size is 10**
- vector1.resize(20);      **//   New size**
- vector2.resize(5);        **//   Can shrink**

## Checking a vector's size

- n = vector1.size();
- if (vector1.empty())

## Access

- vector1[10] = 123;
- x = vector2[3];
- x = vector2[5];        **//   Will fail in undefined way, subscript range is 0 – 4**
- x = vector2.at(i);     **//   Program will abort if i is out of range**

## Automatically increasing size when element is inserted

- vector1.push_back(n);
- Vector class will allocate more memory with *new* operator