CMSC 330: Organization of Programming Languages

Exceptions
Parameter Passing

Preconditions

- Functions often have requirements on their inputs

```
// Return maximum element in A[i..j]
int findMax(int[] A, int i, int j) { ... }
```

  - A is nonempty
  - A isn't null
  - i and j must be nonnegative
  - i and j must be less than A.length
  - i ≤ j (maybe)
- These are called *preconditions*

Dealing with Errors

- What do you do if a precondition isn't met?
- What do you do if something unexpected happens?
  - Try to open a file that doesn't exist
  - Try to write to a full disk

Signaling Errors

- Style 1: Return invalid value

```
// Returns value key maps to, or null if no
// such key in map
Object get(Object key);
```

  - Disadvantages?

Signaling Errors (cont'd)

- Style 2: Return an invalid value and status

```
static int lock_rdev(mdk_rdev_t *rdev) {
  ...
  if (bdev == NULL)
    return -ENOMEM;
  ...
}

// Returns NULL if error and sets global
// variable errno
FILE *fopen(const char *path, const char *mode);
```


Problems with These Approaches

- What if all possible return values are valid?
  - E.g., `findMax` from earlier slide
  - What about errors in a constructor?
- What if client forgets to check for error?
  - No compiler support
- What if client can't handle error?
  - Needs to be dealt with at a higher level
- Poor modularity- exception handling code becomes scattered throughout program
- 1996 Ariane 5 failure classic example of this …