# 15-410

***"...Should we "crash"?..."***

# Errors
# Sep. 23, 2013

**Dave Eckhardt**

**Todd Mowry**

# Outline

**Three kinds of error**

**Important to classify & react appropriately**

# Outline

**Three kinds of error**

- Hmm...
- That's not right...
- Uh-oh...

**Important to classify & react appropriately**

# “New Player” - Take 1

```
// Improve memory locality:
// store players in array; use index, not ptr
struct player players[MAX];
int new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1) {
    /* OH NO!!! */
    MAGIC_BREAK;
  }
  ...
}
```

# “New Player” - Take 2

```
// Improve memory locality:
// store players in array; use index, not ptr
struct player players[MAX];
int new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1) {
    /* OH NO!!! */
    while(1)
      continue;
  }
  ...
}
```

# What's Going On?

**“Out of table slots” - what kind of thing?**

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?

# What's Going On?

**"Out of table slots" - what kind of thing?**

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?
  - Remember: users always want 110%!

**What to do?**

- *Resolve* reasonable issues when possible
  - How to resolve this one?

# “New Player” - Take 3

```
struct player *players;
int playerslots;
int new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1) {
    if ((i = grow_table_and_alloc()) == -1)
      /* OH NO!!! */
      while(1)
        continue;
  }
  ...
}
```

# What's Going On?

**“Out of heap space” - what kind of thing?**

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?

# What's Going On?

**"Out of heap space" - what kind of thing?**

- Should really never happen?
- Might happen sometimes?
- Likely to happen once a day?

**My suggestion**

- "Might happen sometimes"

**What to do?**

- Hard to say what the right thing is for all clients
  - Is it fatal or not?
- Often: pass the buck

# “New Player” - Take 4

```
struct player *players;
int playerslots;
int new_player(int team, int num)
{
  int i;
  if ((i = emptyslot()) == -1) {
    if ((i = grow_table_and_alloc()) == -1)
      return (-1);
  }
  ...
}
```

# “Free Player” - Take 1

```
void free_player(int slot)
{
  assert((slot > 0) && (slot < total_slots));
  struct player *p = &players[slot];
  switch(p->role) {
  case CONTENDER:
    free(p->cstate); break;
  case REFEREE:
    free(p->refstate); break;
  }
  free(p->generic);
  mark_slot_available(slot);
}
```

# What's Wrong?

**There is a sanity-check missing...**

- Probably somebody will make a mistake eventually
- Let's catch it

# “Free Player” - Take 2

```
void free_player(int slot)
{
  assert((slot > 0) && (slot < total_slots));
  struct player *p = &players[slot];
  switch(p->role) {
  case CONTENDER:
    free(p->cstate); break;
  case REFEREE:
    free(p->refstate); break;
  default: return;
  }
  free(p->generic);
  mark_slot_available(slot);
}
```

# All Fixed?

# All Fixed?

**No!**

- The program *has a bug*
  - Maybe the client is passing us random player pointers
  - Maybe we are handing out invalid p->role values
- We happened to catch the bug this time
- We might not catch it every time!
  - A random player pointer might have a “valid” p->role

**The program is *broken***

- Hiding the problem isn't our job
- Hiding the problem isn't even *defensible*

# Should We “Crash”?

**If the program is “broken”, should we “crash”?**

- Often: yes
  - Dumping core allows debugger inspection of the problem
  - Throwing running program into a debugger is probably nicer

# Summary

**Three kinds of error**

- Hmm...
  - Try to *resolve*
- That's not right...
  - Try to *report*
- Uh-oh...
  - Try to *help the developer* find the problem faster