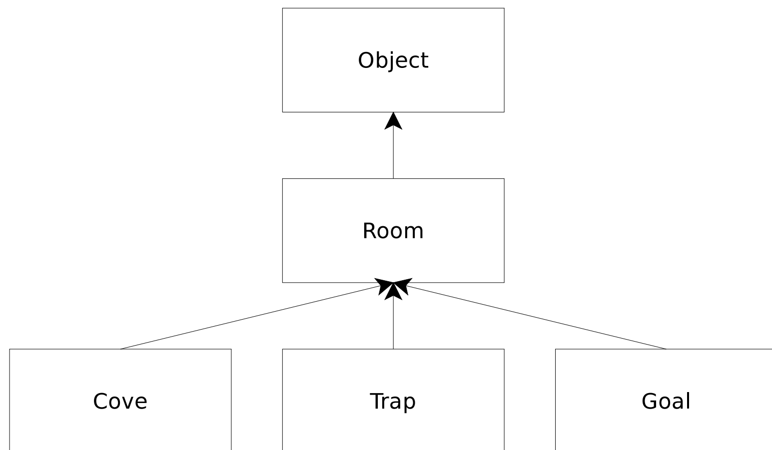
Maze Help Session

CS31

Assignment Overview

- Help Pascal get out of the maze!
- While he earns treasure along the way!
- (Demo)

Room Hierarchy



Room Hierarchy

- Room Fields
  - `int _room_id`
  - `int _neighbors[4]` - IDs of neighboring rooms
- Cove Fields
  - `int _value` - How much money Pascal gets
- Room Methods
  - equals, isSearched, getNeighbors
  - `bool movePascalHere (int* money)`
    - Takes *a pointer to* Pascal's current amount of money
    - Returns true (i.e. 1) if Pascal has reached the goal
- Cove, Trap, and Goal override movePascalHere
- What's `sizeof(Room)`?

Room Hierarchy

- Room Fields
  - `int _room_id`
  - `int _neighbors[4]` - IDs of neighboring rooms
- Cove Fields
  - `int _value` - How much money Pascal gets
- Room Methods
  - equals, isSearched, getNeighbors
  - `bool movePascalHere (int* money)`
    - Takes *a pointer to* Pascal's current amount of money
    - Returns true (i.e. 1) if Pascal has reached the goal
- Cove, Trap, and Goal override movePascalHere
- What's `sizeof(Room)`?

  `= 4 + sizeof(_room_id) + sizeof(_neighbors)`

Room Hierarchy

- Room Fields
  - `int _room_id`
  - `int _neighbors[4]` - IDs of neighboring rooms
- Cove Fields
  - `int _value` - How much money Pascal gets
- Room Methods
  - equals, isSearched, getNeighbors
  - `bool movePascalHere (int* money)`
    - Takes *a pointer to* Pascal's current amount of money
    - Returns true (i.e. 1) if Pascal has reached the goal
- Cove, Trap, and Goal override movePascalHere
- What's `sizeof(Room)`?

```
= 4 + sizeof(_room_id) + sizeof(_neighbors)
= 4 + 4 + 4*4
```

What You Need to Do

- Define the classes and methods for the Room hierarchy (milestone handin)
- Call these methods from inside the search algortihm (milestone handin)
- Implement dynamic memory management (final handin)

What's in a Class?

- Fields
- Methods
- VTBL
- Constructor

Fields

- To access a field, you need to know its offset within the object
- Define a constant like: `queue_num_items = 4`
- If $s0 is a pointer to a Queue instance, access the field like: `lw $s1, queue_num_items($s0)`

Methods

- Really just procedures
- First argument (`$a0`) is a pointer to `this`
- Include the class name in the procedure name for clarity: `queue_enq`

VTBL

- An array that contains the address of each method's procedure
- The very first word in an object is a pointer to the class's VTBL
- MIPS has convenient syntax for declaring an array in the .data section:

  ```
  fib:  .word 1,1,2,3,5,8,13
  ```

- Use with labels when declaring the VTBL:

```
.data
__queue_vtbl: .word object_equals, queue_enq, queue_deq
```

- Define constants for the offsets into the VTBL:

  ```
  queue_vtbl_deq = 8
  ```

Calling a Method

- Look in the VTBL to get the address of the method's procedure
- If `$s0` is pointer to a Queue instance:
  `lw $t0, ($s0)` – pointer to VTBL is now in `$t0`
  `lw $t0, queue_vtbl_enq($t0)` – pointer to enq method is now in `$t0`
- Jump and link to the address in $t0:
  `move $a0, $s0` – Don't forget to pass object in $a0
  `jalr $t0`

Constructor and Initializer

- Name the constructor like this: `construct_queue`
- `this` is passed in `$a0`
- Don't forget to call super-class constructor!
- What if constructor calls a method?

Constructor and Initializer

- VTBL pointer needs to be set first.
- Write an initializer named: `make_queue`
- `this` is passed in `$a0`
- Set the VTBL pointer to point to the Queue VTBL
- Then call `construct_queue`

Instantiating a Queue

- Allocate some memory (e.g. with malloc)
- Call `make_queue`
- Pass pointer to allocated memory in `$a0`
- Remember to free later!

Queue example (note high-level comments!)

```
.data
queue_size          =   16  # public class Queue extends Object{
queue_num_items     =   4   #    int num_items;
queue_head          =   8   #    void* head;
queue_tail          =   12  #    void* tail;
                            #
                            #    Queue();
queue_vtbl_enq      =   4   #    void enq(Object* item);
queue_vtbl_deq      =   8   #    Object* deq();
queue_vtbl_contains =   12  #    bool contains(Object* item);
queue_vtbl_empty    =   16  #    bool empty();
queue_vtbl_print    =   20  #    void print();
                            # }
__queue_vtbl: .word object_equals, queue_enq, queue_deq, queue_c
.text
construct_queue: # ...
make_queue: # ...
# ...
```

Memory Management

- You implement:
  - `void* malloc(int nbytes)`
  - `void free(void* mem)`
  - `void init_heap()`
- Notes
  - `void*` means “address”
  - `malloc` takes number of **bytes**
  - `malloc` must return word-aligned address

For the Milestone

- `init_heap` doesn't need to do anything.
- `malloc` and `free` can call `simple_malloc` and `simple_free`, which we provide.
- Note: `simple_malloc` does not return word-aligned address – make sure to handle this in `malloc`.

You will need to write real memory management for the final handin.

Heap overview

- Heap consists of:
  - Large block of memory
  - A free list pointer
- In the stencil:

```
MEM_SIZE      = 8000
heap:          .word 0:MEM_SIZE
free_lst:      .word   heap
```

The Free List

- Keeps track of free chunks
- Each chunk needs to store:
    - its size
    - a pointer to the next chunk
- Initially the entire heap is just one huge free chunk

malloc

- Iterate the free list and find the **first** chunk whose size can accommodate the amount requested
- General case:
  - Reduce the chunk's size
  - Bite off the last part of the chunk and return a pointer to it
- Edge cases:
  - The chunk is exactly the right size.
  - The chunk is bigger than the size needed, but using it would not leave enough room for the free list metadata.

free

- Put the chunk back on the free list
- Don't worry about fragmentation
- What two things does each free list node need?

free

- Put the chunk back on the free list
- Don't worry about fragmentation
- What two things does each free list node need?
  - Size
  - Pointer to the next chunk

free

- Put the chunk back on the free list
- Don't worry about fragmentation
- What two things does each free list node need?
    - Size
    - Pointer to the next chunk
- How do you find the size?

free

- Put the chunk back on the free list
- Don't worry about fragmentation
- What two things does each free list node need?
  - Size
  - Pointer to the next chunk
- How do you find the size?
  - When you malloc, actually malloc nbytes + 4
  - Use this extra space to store the size
  - Look here when you free

Advice

- Work in words, not bytes
  - malloc needs to return word-aligned memory anyways
  - malloc's argument must still be bytes
  - Just divide by 4 and round up
- Test malloc/free separately from Maze & pay attention to edge cases!