Project 3

Graded

Student

Devin Chen

Total Points

95 / 100 pts

Autograder Score

80.0 / 80.0

Passed Tests

Test compiles (5/5)
Tests Creature == and != overload and display() (4/4)
Tests Cavern default constructor, enterCavern and exitCavern, tallyCategory and CavernReport (4/4)
Tests enterCavern (4/4)
Tests exitCavern (4/4)
Tests tallyCategory (4/4)
Tests CavernReport (7/7)
Tests getLevelSum (4/4)
Tests calculateAvgLevel (5/5)
Tests getTameCount (4/4)
Tests calculateTamePercentage (5/5)
Tests releaseCreaturesBelowLevel (15/15)
Tests releaseCreaturesOfCategory (15/15)

Question 2

Style & Documentation

15 / 20 pts

- ✔ **+ 5 pts** Style
- ✔ **+ 5 pts** Indicates name and date in comment preamble
- **+ 5 pts** Has inline comments where appropriate
- ✔ **+ 5 pts** Has function preambles with @pre, @post, @param, @return where appropriate
- **+ 20 pts** No-Compile Adjustment
- **+ 0 pts** Insufficient submission
- **− 100 pts** Academic integrity
- **− 95 pts** Academic integrity
- **− 90 pts** Academic integrity
- **− 35 pts** Academic integrity

Autograder Results

Test compiles (5/5)

Your program compiles!

Tests Creature == and != overload and display() (4/4)

Your program passed this test.

Tests Cavern default constructor, enterCavern and exitCavern, tallyCategory and CavernReport (4/4)

Your program passed this test.

Tests enterCavern (4/4)

Your program passed this test.

Tests exitCavern (4/4)

Your program passed this test.

Tests tallyCategory (4/4)

Your program passed this test.

Tests CavernReport (7/7)

Your program passed this test.

Tests getLevelSum (4/4)

Your program passed this test.

Tests calculateAvgLevel (5/5)

Your program passed this test.

Tests getTameCount (4/4)

Your program passed this test.

Tests calculateTamePercentage (5/5)

Your program passed this test.

Tests releaseCreaturesBelowLevel (15/15)

Your program passed this test.

Tests releaseCreaturesOfCategory (15/15)

Your program passed this test.

Submitted Files

.vscode/c_cpp_properties.json Download

```
{
  "configurations": [
    {
      "name": "linux-gcc-x64",
      "includePath": [
        "${workspaceFolder}/**"
      ],
      "compilerPath": "/usr/bin/gcc",
      "cStandard": "${default}",
      "cppStandard": "${default}",
      "intelliSenseMode": "linux-gcc-x64",
      "compilerArgs": [
        ""
      ]
    }
  ],
  "version": 4
}
```

.vscode/launch.json Download

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "C/C++ Runner: Debug Session",
      "type": "cppdbg",
      "request": "launch",
      "args": [],
      "stopAtEntry": false,
      "externalConsole": false,
      "cwd": "/mnt/c/Users/devin/Documents/project3-Fobat76",
      "program": "/mnt/c/Users/devin/Documents/project3-Fobat76/build/Debug/outDebug",
      "MIMode": "gdb",
      "miDebuggerPath": "gdb",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ]
    }
  ]
}
```

.vscode/settings.json

Download

```
{
  "files.associations": {
    "atomic": "cpp",
    "cwchar": "cpp",
    "vector": "cpp",
    "exception": "cpp",
    "functional": "cpp",
    "initializer_list": "cpp",
    "iosfwd": "cpp",
    "istream": "cpp",
    "new": "cpp",
    "ostream": "cpp",
    "stdexcept": "cpp",
    "streambuf": "cpp",
    "tuple": "cpp",
    "type_traits": "cpp",
    "utility": "cpp",
    "array": "cpp",
    "bit": "cpp",
    "*.tcc": "cpp",
    "cctype": "cpp",
    "clocale": "cpp",
    "cmath": "cpp",
    "compare": "cpp",
    "concepts": "cpp",
    "cstdarg": "cpp",
    "cstddef": "cpp",
    "cstdint": "cpp",
    "cstdio": "cpp",
    "cstdlib": "cpp",
    "cwctype": "cpp",
    "deque": "cpp",
    "string": "cpp",
    "unordered_map": "cpp",
    "algorithm": "cpp",
    "iterator": "cpp",
    "memory": "cpp",
    "memory_resource": "cpp",
    "numeric": "cpp",
    "random": "cpp",
    "string_view": "cpp",
    "system_error": "cpp",
    "iostream": "cpp",
    "limits": "cpp",
    "numbers": "cpp",
    "typeinfo": "cpp"
  },
  "C_Cpp_Runner.cCompilerPath": "gcc",
  "C_Cpp_Runner.cppCompilerPath": "g++",
```

```
  "C_Cpp_Runner.debuggerPath": "gdb",
  "C_Cpp_Runner.cStandard": "",
  "C_Cpp_Runner.cppStandard": "",
  "C_Cpp_Runner.msvcBatchPath": "",
  "C_Cpp_Runner.useMsvc": false,
  "C_Cpp_Runner.warnings": [
    "-Wall",
    "-Wextra",
    "-Wpedantic",
    "-Wshadow",
    "-Wformat=2",
    "-Wcast-align",
    "-Wconversion",
    "-Wsign-conversion",
    "-Wnull-dereference"
  ],
  "C_Cpp_Runner.msvcWarnings": [
    "/W4",
    "/permissive-",
    "/w14242",
    "/w14287",
    "/w14296",
    "/w14311",
    "/w14826",
    "/w44062",
    "/w44242",
    "/w14905",
    "/w14906",
    "/w14263",
    "/w44265",
    "/w14928"
  ],
  "C_Cpp_Runner.enableWarnings": true,
  "C_Cpp_Runner.warningsAsError": false,
  "C_Cpp_Runner.compilerArgs": [],
  "C_Cpp_Runner.linkerArgs": [],
  "C_Cpp_Runner.includePaths": [],
  "C_Cpp_Runner.includeSearch": [
    "*",
    "**/*"
  ],
  "C_Cpp_Runner.excludeSearch": [
    "**/build",
    "**/build/**",
    "**/.*",
    "**/.*/**",
    "**/.vscode",
    "**/.vscode/**"
  ],
  "C_Cpp_Runner.useAddressSanitizer": false,
  "C_Cpp_Runner.useUndefinedSanitizer": false,
  "C_Cpp_Runner.useLeakSanitizer": false,
```

```
  "C_Cpp_Runner.showCompilationTime": false,
  "C_Cpp_Runner.useLinkTimeOptimization": false,
  "C_Cpp_Runner.msvcSecureNoWarnings": false
}
```

ArrayBag.cpp

Download

```cpp
/*
ArrayBag interface for term project
CSCI 235 Spring 2024
*/


#include "ArrayBag.hpp"

/** default constructor**/
template<class ItemType>
ArrayBag<ItemType>::ArrayBag(): item_count_(0)
{
} // end default constructor

/**
 @return item_count_ : the current size of the bag
 **/
template<class ItemType>
int ArrayBag<ItemType>::getCurrentSize() const
{
	return item_count_;
} // end getCurrentSize

/**
 @return true if item_count_ == 0, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::isEmpty() const
{
	return item_count_ == 0;
} // end isEmpty

/**
 @return true if new_entry was successfully added to items_, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& new_entry)
{
	bool has_room = (item_count_ < DEFAULT_CAPACITY);
	if (has_room)
	{
		items_[item_count_] = new_entry;
		item_count_++;
	return true;
	} // end if

	return false;
} // end add

```

```
/**
 @return true if an_entry was successfully removed from items_, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& an_entry)
{
  int found_index = getIndexOf(an_entry);
      bool can_remove = !isEmpty() && (found_index > -1);
      if (can_remove)
      {
            item_count_--;
            items_[found_index] = items_[item_count_];
      }  // end if

      return can_remove;
}  // end remove

/**
 @post item_count_ == 0
 **/
template<class ItemType>
void ArrayBag<ItemType>::clear()
{
      item_count_ = 0;
}  // end clear

/**
 @return the number of times an_entry is found in items_
 **/
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& an_entry) const
{
  int frequency = 0;
  int curr_index = 0;       // Current array index
  while (curr_index < item_count_)
  {
    if (items_[curr_index] == an_entry)
    {
      frequency++;
    }  // end if

    curr_index++;          // Increment to next entry
  }  // end while

  return frequency;
}  // end getFrequencyOf

/**
 @return true if an_entry is found in items_, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& an_entry) const
```

```
{
        return getIndexOf(an_entry) > -1;
}  // end contains

// ********* PRIVATE METHODS **************//

/**
        @param target to be found in items_
        @return either the index target in the array items_ or -1,
        if the array does not containthe target.
 **/
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
{
        bool found = false;
  int result = -1;
  int search_index = 0;
   // If the bag is empty, item_count_ is zero, so loop is skipped
   while (!found && (search_index < item_count_))
   {

      if (items_[search_index] == target)
      {
         found = true;
         result = search_index;
      }
      else
      {
         search_index++;
      }  // end if
   }  // end while

   return result;
}  // end getIndexOf


```

ArrayBag.hpp

Download

```cpp
/*
ArrayBag interface for term project
CSCI 235 Spring 2024
*/

#ifndef ARRAY_BAG_
#define ARRAY_BAG_
#include <iostream>
#include <vector>

template <class ItemType>
class ArrayBag
{

  public:
  /** default constructor**/
  ArrayBag();

  /**
     @return item_count_ : the current size of the bag
  **/
  int getCurrentSize() const;

  /**
     @return true if item_count_ == 0, false otherwise
  **/
  bool isEmpty() const;

  /**
     @return true if new_entry was successfully added to items_, false otherwise
  **/
  bool add(const ItemType &new_entry);

  /**
     @return true if an_entry was successfully removed from items_, false otherwise
    **/
  bool remove(const ItemType &an_entry);

  /**
     @post item_count_ == 0
    **/
  void clear();

  /**
     @return true if an_entry is found in items_, false otherwise
    **/
  bool contains(const ItemType &an_entry) const;

  /**
```

```
    @return the number of times an_entry is found in items_
  **/
  int getFrequencyOf(const ItemType &an_entry) const;

  protected:
  static const int DEFAULT_CAPACITY = 100; //max size of items_ at 100 by default for this project
  ItemType items_[DEFAULT_CAPACITY];      // Array of bag items
  int item_count_;                        // Current count of bag items

  /**
     @param target to be found in items_
    @return either the index target in the array items_ or -1,
    if the array does not contain the target.
    **/
  int getIndexOf(const ItemType &target) const;

}; // end ArrayBag

#include "ArrayBag.cpp"
#endif

```

Cavern.cpp Download

```cpp
/**
 * @file Cavern.cpp
 * @author Devin Chen
 * @brief Cavern Class
 * @date 3/6/2024
*/
#include "Cavern.hpp"

/**
 * Default constructor.
 * Default-initializes all private members.
*/
Cavern::Cavern(){
   level_sum_ =0;
   tame_count_=0;
}
/**
* @param   :   A reference to a Creature entering the Cavern
* @post    :   If the given Creature is not already in the Cavern, add Creature to the Cavern and updates the level sum and the tame Creature count if the creature is tame.
* @return  :   returns true if a Creature was successfully added to the Cavern, false otherwise
          :   Hint: Use the above definition of equality will help determine if a Creature is already in the Cavern
**/
bool Cavern::enterCavern(const Creature &new_creature_added){
if(contains(new_creature_added)){
   return false;
   }
else{
   add(new_creature_added);
   level_sum_ += new_creature_added.getLevel();
   if(new_creature_added.isTame()){
      tame_count_++;
      }
   return true;
   }
}
/**
* @param   :   A  reference to a Creature leaving the Cavern
* @return  :  returns true if a creature was successfully removed from the Cavern (i.e. items_), false otherwise
* @post    :    removes the creature from the Cavern and updates the level sum.
* If the Creature is tame it also updates the tame count.
**/
bool Cavern::exitCavern(const Creature &new_creature_leaving){
   if(remove(new_creature_leaving)){
      level_sum_ -= new_creature_leaving.getLevel();
      if(new_creature_leaving.isTame()){
         tame_count_--;
```

```
        }
        return true;
    }
    return false;
}
/**
* @return   :  The integer level count of all the creatures currently in the Cavern
**/
int Cavern::getLevelSum()const{
    return level_sum_;
}
/**
* @return   :  The integer level count of all the creatures currently in the Cavern
**/
int Cavern::calculateAvgLevel()const{
    if(getCurrentSize() == 0){
        return 0;
    }
    return round(static_cast<double>(level_sum_) / getCurrentSize());
}
/**
* @return   :  The integer count of tame Creatures in the Cavern
**/
int Cavern::getTameCount()const{
    return tame_count_;
}
/**
* @return   :  The percentage (double) of all the tame creatures in the Cavern
* @post     :   Computes the percentage of tame creatures in the Cavern rounded up to 2 decimal places.
**/
double Cavern::calculateTamePercentage() const {
    if (getCurrentSize() == 0) {
        return 0;
    }
    double percentage = static_cast<double>(tame_count_) / static_cast<double>(getCurrentSize()) * 100;
    return std::ceil(percentage * 100) / 100;
}
/**
* @param  :   A reference to a string representing a creature Category with value in ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"]
* @return  :  An integer tally of the number of creatures in the Cavern of the given category.
*If the argument string does not match one of the expected category values, the tally is zero.
*NOTE: no pre-processing of the input string necessary, only uppercase input will match.
**/
int Cavern::tallyCategory(const std::string &Category)const{
int tally = 0;
for(int i =0; i < getCurrentSize(); i ++){
    if(items_[i].getCategory() == Category){
        tally ++;
    }
```

```
}
return tally;
}
/**
* @param  :   An integer representing the level threshold of the creatures to be removed from the Cavern, with default value 0
* @post   :   Removes all creatures from the Cavern whose level is less than the given level. If no level is given, removes all creatures from the Cavern. Ignore negative input.
* @return :   The number of creatures removed from the Cavern
*/
int Cavern::releaseCreaturesBelowLevel(int threshold) {
    int numremoved = 0;
    if(threshold == 0){
        numremoved = getCurrentSize();
        clear();
        level_sum_ =0;
        tame_count_ =0;
    }
    if(threshold > 0){
        for(int i = 0; i < getCurrentSize(); i++){
            if(items_[i].getLevel() < threshold){
                exitCavern(items_[i]);
                numremoved ++;
                i--;
            }
        }

    }
    return numremoved;
}
/**
* @param  : A reference to a string representing a creature Category with a value in ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"], or default value "ALL" if no category is given
* @post   : Removes all creatures from the Cavern whose category matches the given category. If no category is given, removes all creatures from the Cavern.
* @return : The number of creatures removed from the Cavern
*NOTE: no pre-processing of the input string necessary, only uppercase input will match. If the input string does not match one of the expected category values, do not remove any creatures.
*/
int Cavern::releaseCreaturesOfCategory(const std::string &value){
    int numremoved = 0;
    if(value == "ALL"){
            for(int i = 0; i < getCurrentSize(); i++){
        exitCavern(items_[i]);
        numremoved++;
            }
    }
    else{
        for(int i =0; i < getCurrentSize(); i++){
            if(items_[i].getCategory() == value){
                exitCavern(items_[i]);
                numremoved++;
```

```
            }
        }
    }
    return numremoved;
    }
/**
* @post    : Outputs a report of the creatures currently in the Cavern in the form:
    "UNKNOWN: [x]\nUNDEAD: [x]\nMYSTICAL: [x]\nALIEN: [x]\n\nAVERAGE LEVEL: [x]\nTAME:[x]%\n"
    Note that the average level should be rounded to the NEAREST integer, and the percentage of tame creatures in the Cavern should be rounded to 2 decimal places.

Example output:
UNKNOWN: 1
UNDEAD: 3
MYSTICAL: 2
ALIEN: 1

AVERAGE LEVEL: 5
TAME: 85.72%
*/
void Cavern::cavernReport()const {
std::cout << "UNKNOWN: " << tallyCategory("UNKNOWN") << std::endl;
std::cout << "UNDEAD: " << tallyCategory("UNDEAD") << std::endl;
std::cout << "MYSTICAL: " << tallyCategory("MYSTICAL") << std::endl;
std::cout << "ALIEN: " << tallyCategory("ALIEN") << std::endl << std::endl;
std::cout << "AVERAGE LEVEL: " << calculateAvgLevel() << std::endl;
std::cout << "TAME: " << calculateTamePercentage() << "%" << std::endl;
}
```

Cavern.hpp

Download

```
/**
 * @file Cavern.hpp
 * @author Devin Chen
 * @brief Cavern Class
 * @date 3/6/2024
*/
#include "ArrayBag.hpp"
#include "Creature.hpp"
#include <cmath>
#pragma once
class Cavern:public ArrayBag<Creature> {
    public:
/**
 * Default constructor.
 * Default-initializes all private members.
*/
        Cavern();
/**
* @param   :   A reference to a Creature entering the Cavern
* @post    :   If the given Creature is not already in the Cavern, add Creature to the Cavern and updates the level sum and the tame Creature count if the creature is tame.
* @return  :   returns true if a Creature was successfully added to the Cavern, false otherwise
          :   Hint: Use the above definition of equality will help determine if a Creature is already in the Cavern
**/
        bool enterCavern(const Creature& creature);
/**
* @param   :   A  reference to a Creature leaving the Cavern
* @return  :  returns true if a creature was successfully removed from the Cavern (i.e. items_), false otherwise
* @post    :    removes the creature from the Cavern and updates the level sum.
* If the Creature is tame it also updates the tame count.
**/
        bool exitCavern(const Creature& creature);

/**
* @return   :  The integer level count of all the creatures currently in the Cavern
**/
        int getLevelSum() const;
/**
* @return   :  The integer level count of all the creatures currently in the Cavern
**/
        int calculateAvgLevel() const;
/**
* @return   :  The integer count of tame Creatures in the Cavern
**/
        int getTameCount() const;
/**
* @return   :  The percentage (double) of all the tame creatures in the Cavern
```

```
* @post    :   Computes the percentage of tame creatures in the Cavern rounded up to 2 decimal places.
**/
        double calculateTamePercentage() const;
/**
* @param  :   A reference to a string representing a creature Category with value in ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"]
* @return  :  An integer tally of the number of creatures in the Cavern of the given category.
*If the argument string does not match one of the expected category values, the tally is zero.
*NOTE: no pre-processing of the input string necessary, only uppercase input will match.
**/
        int tallyCategory(const std::string& category) const;
/**
* @param  :   An integer representing the level threshold of the creatures to be removed from the Cavern, with default value 0
* @post   :   Removes all creatures from the Cavern whose level is less than the given level. If no level is given, removes all creatures from the Cavern. Ignore negative input.
* @return :   The number of creatures removed from the Cavern
*/
        int releaseCreaturesBelowLevel(const int threshold = 0);
/**
* @param  : A reference to a string representing a creature Category with a value in ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"], or default value "ALL" if no category is given
* @post   : Removes all creatures from the Cavern whose category matches the given category. If no category is given, removes all creatures from the Cavern.
* @return : The number of creatures removed from the Cavern
*NOTE: no pre-processing of the input string necessary, only uppercase input will match. If the input string does not match one of the expected category values, do not remove any creatures.
*/
        int releaseCreaturesOfCategory(const std::string& category = "ALL");
/**
* @post    : Outputs a report of the creatures currently in the Cavern in the form:
    "UNKNOWN: [x]\nUNDEAD: [x]\nMYSTICAL: [x]\nALIEN: [x]\n\nAVERAGE LEVEL: [x]\nTAME:[x]%\n"
    Note that the average level should be rounded to the NEAREST integer, and the percentage of tame creatures in the Cavern should be rounded to 2 decimal places.

Example output:
UNKNOWN: 1
UNDEAD: 3
MYSTICAL: 2
ALIEN: 1

AVERAGE LEVEL: 5
TAME: 85.72%
*/
        void cavernReport() const;
    private:
        int level_sum_;
        int tame_count_;
};
```

Creature.cpp

Download

```cpp
/*
CSCI235 Spring 2024
Project 1 - Creature Class
Georgina Woo
Nov 13 2023
Creature.cpp defines the constructors and private and public function implementation of the Creature class
*/

#include "Creature.hpp"

/**
    Default constructor.
    Default-initializes all private members.
    Default creature name: "NAMELESS".
    Booleans are default-initialized to False.
    Default enum value: UNKNOWN
    Default Hitpoints and Level: 1.
*/
Creature::Creature(): name_{"NAMELESS"},  category_{UNKNOWN}, hitpoints_{1}, level_{1}, tame_{false}
{

}

/**
    Parameterized constructor.
    @param      : A reference to the name of the creature (a string). Set the creature's name to NAMELESS if the provided string contains non-alphabetic characters.
    @param      : The category of the creature (a Category enum) with default value UNKNOWN
    @param      : The creature's hitpoints (an integer) , with default value 1 if not provided, or if the value provided is 0 or negative
    @param      : The creature's level (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
    @param      : A flag indicating whether the creature is tame, with default value False
    @post       : The private members are set to the values of the corresponding parameters.
    Hint: Notice the default arguments in the parameterized constructor.
*/
Creature::Creature(const std::string& name, Category category, int hitpoints, int level, bool tame): category_{category}
{
    if(!setName(name))
    {
        name_ = "NAMELESS";
    }

    if(!setHitpoints(hitpoints))
    {
        hitpoints_ = 1;
```

```
    }
    if(!setLevel(level))
    {
        level_ = 1;
    }
    tame_ = tame;

}

/**
    @param : the name of the Creature, a reference to string
    @post  : sets the Creature's name to the value of the parameter in UPPERCASE.
             (convert any lowercase character to uppercase)
             Only alphabetical characters are allowed.
           : If the input contains non-alphabetic characters, do nothing.
    @return : true if the name was set, false otherwise
*/
bool Creature::setName(const std::string& name)
{
    if (name.length() == 0)
    {
        return false;
    }
    else
    {
        std::string nameUpper = name;
        for (int i = 0; i < name.length(); i++)
        {
            if (!isalpha(name[i]))
            {
                return false;
            }
            else
            {
                nameUpper[i] = toupper(name[i]);
            }
        }
        name_ = nameUpper;
        return true;
    }
}

/**
     @return : the name of the Creature
*/
std::string Creature::getName() const
{
    return name_;
}


/**
```

```
    @param  : the category of the Creature (an enum)
    @post   : sets the Creature's category to the value of the parameter
*/
void Creature::setCategory(const Category& category)
{
    category_ = category;
}


/**
     @return : the category of the Creature (in string form)
*/
std::string Creature::getCategory() const
{
    switch(category_)
    {
        case UNDEAD:
            return "UNDEAD";
        case MYSTICAL:
            return "MYSTICAL";
        case ALIEN:
            return "ALIEN";
        default:
            return "UNKNOWN";
    }
}

/**
    @param  : an integer that represents the creature's hitpoints
    @pre    : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid input)
    @post   : sets the hitpoints private member to the value of the parameter
    @return : true if the hitpoints were set, false otherwise
*/
bool Creature::setHitpoints(const int& hitpoints)
{
    if (hitpoints > 0)
    {
        hitpoints_ = hitpoints;
        return true;
    }
    else
    {
        return false;
    }
}


/**
     @return : the value stored in hitpoints_
*/
int Creature::getHitpoints() const
```

```
{
    return hitpoints_;
}

/**
    @param  : an integer level
    @pre    : level >  0 : Characters cannot have 0 or negative levels (do nothing for invalid input)
    @post   : sets the level private member to the value of the parameter
    @return : true if the level was set, false otherwise
*/
bool Creature::setLevel(const int& level)
{
    if (level > 0)
    {
        level_ = level;
        return true;
    }
    else
    {
        return false;
    }
}


/**
     @return : the value stored in level_
*/
int Creature::getLevel() const
{
    return level_;
}


/**
    @param  : a boolean value
    @post   : sets the tame flag to the value of the parameter
*/
void Creature::setTame(const bool& tame)
{
    tame_ = tame;
}


/**
    @return true if the creature is tame, false otherwise
    Note: this is an accessor function and must follow the same convention as all accessor functions even if it is not called getTame
*/
bool Creature::isTame() const
{
    return tame_;
}
```

```

/**
    @post     : displays Creature data in the form:
    "[NAME]\n
    Category: [CATEGORY]\n
    Level: [LEVEL]\n
    Hitpoints: [Hitpoints]\n
    Tame: [TRUE/FALSE]"
*/
void Creature::display() const
{
    std::cout << name_ << std::endl;
    std::cout << "Category: " << getCategory() << std::endl;
    std::cout << "Level: " << level_ << std::endl;
    std::cout << "Hitpoints: " << hitpoints_ << std::endl;
    std::cout << "Tame: " << (tame_ ? "TRUE" : "FALSE") << std::endl;
}

/**
 * @param      : a const reference to the right hand side of the == operator.
 * @return     : Returns true if the right hand side creature is "equal", false otherwise.
 * Two creatures are equal if they have the same name, same category, same level, and if they're either both tame or both not
 * NOTE: By this definition, only the aforementioned subset of the creature's attributes must be equal for two creatures to be deemed "equal".

* Example: In order for creature1 to be == to creature2 we only need:
* The same name
* The same category
* The same level
* They must either be both tame or both not
*/
bool Creature::operator== (const Creature &reference)const{
    if(reference.name_ == name_ && reference.level_ == level_ && reference.category_ == category_ && reference.tame_ == tame_){
        return true;
    }
    return false;
}

/**
 * @param      : a const reference to the right hand side of the != operator.
 * @return     : Returns true if the right hand side creature is NOT "equal" (!=), false
 * otherwise. Two creatures are NOT equal if any of their name, category or level are not equal, or if one is tame and the other is not.
 * NOTE: By this definition, one or more of the aforementioned subset of the creature's attributes only must be different for two creatures to be deemed "NOT equal".
*/
bool Creature::operator!= (const Creature &reference)const{
    if(reference.name_ == name_ && reference.level_ == level_ && reference.category_ == category_ && reference.tame_ == tame_){
        return false;
```

```
    }
    return true;
}
```

Creature.hpp

Download

```cpp
/*
CSCI235 Spring 2024
Project 1 - Creature Class
Georgina Woo
Nov 13 2023
Creature.hpp declares the Creature class along with its private and public members
*/
#ifndef CREATURE_HPP_
#define CREATURE_HPP_
#include <iostream>
#include <string>
#include <cctype>


class Creature
{
    public:
        enum Category {UNKNOWN, UNDEAD, MYSTICAL, ALIEN};
        /**
            Default constructor.
            Default-initializes all private members.
            Default creature name: "NAMELESS".
            Booleans are default-initialized to False.
            Default enum value: UNKNOWN
            Default Hitpoints and Level: 1.
        */
        Creature();

        /**
            Parameterized constructor.
            @param      : The name of the creature (a string)
            @param      : The category of the creature (a Category enum) with default value UNKNOWN
            @param      : The creature's hitpoints (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
            @param      : The creature's level (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
            @param      : A flag indicating whether the creature is tame, with default value False
            @post       : The private members are set to the values of the corresponding parameters.
            Hint: Notice the default arguments in the parameterized constructor.
        */
        Creature(const std::string& name, Category category = UNKNOWN, int hitpoints = 1, int level = 1, bool tame = false);

        /**
             @param : the name of the Creature, a string
             @post  : sets the Creature's name to the value of the parameter in UPPERCASE (convert any lowercase character to upppercase
                      Only alphabetical characters are allowed.
                    : If the input contains non-alphabetic characters, do nothing.
```

```
        @return : true if the name was set, false otherwise
    */
    bool setName(const std::string& name);

    /**
         @return : the name of the Creature
    */
    std::string getName() const;


    /**
        @param  : the category of the Creature (an enum)
        @post   : sets the Creature's category to the value of the parameter
    */
    void setCategory(const Category& category);


    /**
         @return : the race of the Creature (in string form)
    */
    std::string getCategory() const;

    /**
        @param  : an integer that represents the creature's hitpoints
        @pre    : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid input)
        @post   : sets the hitpoints private member to the value of the parameter
        @return : true if the hitpoints were set, false otherwise
    */
    bool setHitpoints(const int& hitpoints);


    /**
         @return : the value stored in hitpoints_
    */
    int getHitpoints() const;

    /**
        @param  : an integer level
        @pre    : level > 0 : Creatures cannot have 0 or negative levels (do nothing for invalid input)
        @post   : sets the level private member to the value of the parameter
        @return : true if the level was set, false otherwise
    */
    bool setLevel(const int& level);


    /**
         @return : the value stored in level_
    */
    int getLevel() const;


```

```
    /**
       @param  : a boolean value
       @post   : sets the tame flag to the value of the parameter
    */
    void setTame(const bool& tame);


    /**
       @return true if the Creature is tame, false otherwise
       Note: this is an accessor function and must follow the same convention as all accessor functions even if it is not called getTame
    */
    bool isTame() const;

     /**
       @post     : displays Creature data in the form:
       "[NAME]\n
       Category: [CATEGORY]\n
       Level: [LEVEL]\n
       Hitpoints: [Hitpoints]\n
       Tame: [TRUE/FALSE]"
    */
    void display() const;

    /**
     * @param      : a const reference to the right hand side of the == operator.
     * @return     : Returns true if the right hand side creature is "equal", false otherwise.
     * Two creatures are equal if they have the same name, same category, same level, and if they're either both tame or both not
     * NOTE: By this definition, only the aforementioned subset of the creature's attributes must be equal for two creatures to be deemed "equal".

     * Example: In order for creature1 to be == to creature2 we only need:
     * The same name
     * The same category
     * The same level
     * They must either be both tame or both not
    */
   bool operator== (const Creature &reference)const;

    /**
     * @param      : a const reference to the right hand side of the != operator.
     * @return     : Returns true if the right hand side creature is NOT "equal" (!=), false
     * otherwise. Two creatures are NOT equal if any of their name, category or level are not equal, or if one is tame and the other is not.
     * NOTE: By this definition, one or more of the aforementioned subset of the creature's attributes only must be different for two creatures to be deemed "NOT equal".
    */
   bool operator!= (const Creature &reference)const;

  private:
    // The name of the creature (a string in UPPERCASE)
```

```
    std::string name_;
    // The category of the creature (an enum)
    Category category_;
    // The creature's hitpoints (a non-zero, non-negative integer)
    int hitpoints_;
    // The creature's level (a non-zero, non-negative integer)
    int level_;
    // A flag indicating whether the creature is tame
    bool tame_;

};

#endif
```

Makefile Download

```
CXX = g++
CXXFLAGS = -std=c++17 -g -Wall -O2

PROG ?= main
OBJS = Creature.o main.o Cavern.o

all: $(PROG)

.cpp.o:
	$(CXX) $(CXXFLAGS) -c -o $@ $<

$(PROG): $(OBJS)
	$(CXX) $(CXXFLAGS) -o $@ $(OBJS)

clean:
	rm -rf $(EXEC) *.o *.out main

rebuild: clean all

```

README.md Download

```
[![Review Assignment Due Date](https://classroom.github.com/assets/deadline-readme-button-24ddc0f5d75046c5622901739e7c5dd533143b0c8e959d652212380cedb1ea36.svg)](https://classroom.github.com/a/tgc2wzn0)
# Project3



The project specification can be found on Blackboard
```