Project 1

● Graded

Student

Devin Chen

Total Points

100 / 100 pts

Autograder Score

80.0 / 80.0

Passed Tests

Test compiles (5/5)
Tests Creature class default constructor (10/10)
Tests Creature class parameterized constructor with all arguments given (10/10)
Tests Character class parameterized constructor with default arguments (10/10)
Tests Creature class mutator functions (25/25)
Tests Creature class display function (5/5)
Checks that test.cpp tests Creature functions (10/10)
Test for read-only functions and parameters (5/5)

Question 2

Style & Documentation

20 / 20 pts

- ✓ **+ 5 pts** Style
- ✓ **+ 5 pts** Indicates name and and description in comment preamble at top of file
- ✓ **+ 10 pts** Has function preambles with @pre, @post, @param, @return where appropriate
- **+ 20 pts** No-Compile Adjustment
- **+ 0 pts** Insufficient submission

Autograder Results

Test compiles (5/5)

Your program compiles!

Tests Creature class default constructor (10/10)

Your program passed.

Tests Creature class parameterized constructor with all arguments given (10/10)

Your program passed.

Tests Character class parameterized constructor with default arguments (10/10)

Your program passed.

Tests Creature class mutator functions (25/25)

Your program passed.

Tests Creature class display function (5/5)

Your program passed.

Checks that test.cpp tests Creature functions (10/10)

test.cpp compiles!

Your program is testing the Creature class appropriately.

Test for read-only functions and parameters (5/5)

Your program has functions and parameters as read-only (const) where appropriate.

Submitted Files

.gitignore Download

```
.DS_Store
.vscode
*.log


```

Creature.cpp
Download

```cpp
/**
 * @file Creature.hpp
 * @author Devin Chen
 * @brief Creature Class
 * @date 1/20/2024
*/


#include "Creature.hpp"

/**
 * Defaut 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},is_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. The
name is converted to UPPERCASE if it consists of alphabetical characters only, otherwise it is set to
NAMELESS.
*/
Creature::Creature(const std::string &new_name, Category new_category, int new_hitpoint, int
new_level, bool new_tame){
    if(!setName(new_name)) {
        name_ = "NAMELESS";
    }
    setCategory(new_category);
    if(!setLevel(new_level)){
        level_ = 1;
    }
    if(!setHitpoints(new_hitpoint)){
        hitpoints_ = 1;
    }
    setTame(new_tame);
}
```

```cpp


/**
 * @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 &new_name){
    for(int i = 0; i < new_name.length(); i++){
        if (!isalpha(new_name[i])){
            return false;
        }
    };
    std::string tempname;
    for (int i = 0; i < new_name.length(); i++){
        tempname += toupper(new_name[i]);
    }
    name_ = tempname;
    return true;
};

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

/**
 * @param  : a reference to Category, the category of the Creature (an enum)
 * @post   : sets the Creature's category to the value of the parameter
 * : If the given category was invalid, set category_ to UNKNOWN.
*/
void Creature::setCategory(const Category &new_category){
    if(new_category >= UNKNOWN && new_category <= ALIEN) {
        category_ = new_category;
    }
    else{
        category_ = UNKNOWN;
    }
}
/**
 * @return : the category of the Creature (in string form)
*/
std::string Creature::getCategory() const{
    switch(category_){
        case UNKNOWN:
            return "UNKNOWN";
        case UNDEAD:
            return "UNDEAD";
        case MYSTICAL:
            return "MYSTICAL";
        case ALIEN:
            return "ALIEN";
    }
}

/**
 * @param  : a reference to integer that represents the creature's hitpoints
 * @pre    : hitpoints >= 0 : Characters cannot have 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 &new_hitpoint){
    if (new_hitpoint <= 0){
        return false;
    }
    hitpoints_ = new_hitpoint;
    return true;
}

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

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

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

```

```cpp
/**
 * @param  : a reference to boolean value
 * @post   : sets the tame flag to the value of the parameter
*/
void Creature::setTame(const bool &new_tame){
    is_tame_ = new_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 is_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_ << "\n";
    std::cout << "Category: " << getCategory() << "\n";
    std::cout << "Level: " << level_ << "\n";
    std::cout << "Hitpoints: " << hitpoints_ << "\n";
    if(is_tame_){
        std::cout << "Tame: TRUE";
    }
        else {
        std::cout << "Tame: FALSE";
    }
}

```

Creature.hpp Download

```cpp
/**
 * @file Creature.hpp
 * @author Devin Chen
 * @brief Creature Class
 * @date 1/20/2024
*/

#pragma once
#include <iostream>
#include <string>
#include <cctype>

class Creature{
public:
    enum Category {UNKNOWN, UNDEAD, MYSTICAL, ALIEN};

private:
    std::string name_;
    Category category_;
    int hitpoints_;
    int level_;
    bool is_tame_;

public:

/**
 * Defaut 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      : 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. The name is converted to UPPERCASE if it consists of alphabetical characters only, otherwise it is set to NAMELESS.
*/
```

```cpp
Creature(const std::string &new_name, Category new_category = UNKNOWN, int new_hitpoint = 1,
int new_level = 1 , bool new_tame = false);

/**
 * @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 setName(const std::string &new_name);

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

/**
 * @param  : a reference to Category, the category of the Creature (an enum)
 * @post   : sets the Creature's category to the value of the parameter
 * : If the given category was invalid, set category_ to UNKNOWN.
*/
void setCategory(const Category &new_category);

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

/**
 * @param  : a reference to integer that represents the creature's hitpoints
 * @pre    : hitpoints >= 0 : Characters cannot have 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 &new_hitpoint);


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

/**
 * @param  : a reference to integer level
 * @pre    : level >= 0 : Characters cannot have a negative level
 * @post   : sets the level private member to the value of the parameter
 * (do nothing for invalid input)
 * @return : true if the level was set, false otherwise
*/
```

```
bool setLevel(const int &new_level);

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

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

Makefile Download

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

PROG ?= main
OBJS = Creature.o test.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

```markdown
[![Review Assignment Due Date](https://classroom.github.com/assets/deadline-readme-button-24ddc0f5d75046c5622901739e7c5dd533143b0c8e959d652212380cedb1ea36.svg)](https://classroom.github.com/a/kuHzDhWw)
# Project1

The project specification can be found on Blackboard
```

test.cpp Download

```cpp
#include "Creature.hpp"

int main(){

    Creature dragon;
    dragon.setHitpoints(10);
    dragon.setLevel(5);
    dragon.setTame(true);
    dragon.display();

    Creature worm("wormy",Creature::Category::MYSTICAL, 3,2);
    worm.setTame(true);
    worm.display();
}
```