

## 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

```
1  {
2    "configurations": [
3      {
4        "name": "linux-gcc-x64",
5        "includePath": [
6          "${workspaceFolder}/**"
7        ],
8        "compilerPath": "/usr/bin/gcc",
9        "cStandard": "${default}",
10       "cppStandard": "${default}",
11       "intelliSenseMode": "linux-gcc-x64",
12       "compilerArgs": [
13         ""
14       ]
15     }
16   ],
17   "version": 4
18 }
```

## ▼ .vscode/launch.json

 Download

```
1  {
2    "version": "0.2.0",
3    "configurations": [
4      {
5        "name": "C/C++ Runner: Debug Session",
6        "type": "cppdbg",
7        "request": "launch",
8        "args": [],
9        "stopAtEntry": false,
10       "externalConsole": false,
11       "cwd": "/mnt/c/Users/devin/Documents/project3-Fobat76",
12       "program": "/mnt/c/Users/devin/Documents/project3-Fobat76/build/Debug/outDebug",
13       "MIMode": "gdb",
14       "miDebuggerPath": "gdb",
15       "setupCommands": [
16         {
17           "description": "Enable pretty-printing for gdb",
18           "text": "-enable-pretty-printing",
19           "ignoreFailures": true
20         }
21       ]
22     }
23   ]
24 }
```

```
1  {
2  "files.associations": {
3    "atomic": "cpp",
4    "cwchar": "cpp",
5    "vector": "cpp",
6    "exception": "cpp",
7    "functional": "cpp",
8    "initializer_list": "cpp",
9    "iosfwd": "cpp",
10   "istream": "cpp",
11   "new": "cpp",
12   "ostream": "cpp",
13   "stdexcept": "cpp",
14   "streambuf": "cpp",
15   "tuple": "cpp",
16   "type_traits": "cpp",
17   "utility": "cpp",
18   "array": "cpp",
19   "bit": "cpp",
20   "*.tcc": "cpp",
21   "cctype": "cpp",
22   "clocale": "cpp",
23   "cmath": "cpp",
24   "compare": "cpp",
25   "concepts": "cpp",
26   "cstdarg": "cpp",
27   "cstddef": "cpp",
28   "cstdint": "cpp",
29   "cstdio": "cpp",
30   "cstdlib": "cpp",
31   "cwctype": "cpp",
32   "deque": "cpp",
33   "string": "cpp",
34   "unordered_map": "cpp",
35   "algorithm": "cpp",
36   "iterator": "cpp",
37   "memory": "cpp",
38   "memory_resource": "cpp",
39   "numeric": "cpp",
40   "random": "cpp",
41   "string_view": "cpp",
42   "system_error": "cpp",
43   "iostream": "cpp",
44   "limits": "cpp",
45   "numbers": "cpp",
46   "typeinfo": "cpp"
47  },
48  "C_Cpp_Runner.cCompilerPath": "gcc",
49  "C_Cpp_Runner.cppCompilerPath": "g++",
```

```
50 "C_Cpp_Runner.debuggerPath": "gdb",
51 "C_Cpp_Runner.cStandard": "",
52 "C_Cpp_Runner.cppStandard": "",
53 "C_Cpp_Runner.msvcBatchPath": "",
54 "C_Cpp_Runner.useMsvc": false,
55 "C_Cpp_Runner.warnings": [
56     "-Wall",
57     "-Wextra",
58     "-Wpedantic",
59     "-Wshadow",
60     "-Wformat=2",
61     "-Wcast-align",
62     "-Wconversion",
63     "-Wsign-conversion",
64     "-Wnull-dereference"
65 ],
66 "C_Cpp_Runner.msvcWarnings": [
67     "/W4",
68     "/permissive-",
69     "/w14242",
70     "/w14287",
71     "/w14296",
72     "/w14311",
73     "/w14826",
74     "/w44062",
75     "/w44242",
76     "/w14905",
77     "/w14906",
78     "/w14263",
79     "/w44265",
80     "/w14928"
81 ],
82 "C_Cpp_Runner.enableWarnings": true,
83 "C_Cpp_Runner.warningsAsError": false,
84 "C_Cpp_Runner.compilerArgs": [],
85 "C_Cpp_Runner.linkerArgs": [],
86 "C_Cpp_Runner.includePaths": [],
87 "C_Cpp_Runner.includeSearch": [
88     "**",
89     "**/*"
90 ],
91 "C_Cpp_Runner.excludeSearch": [
92     "**/build",
93     "**/build/**",
94     "**/*.*",
95     "**/*.*",
96     "**/.vscode",
97     "**/.vscode/**"
98 ],
99 "C_Cpp_Runner.useAddressSanitizer": false,
100 "C_Cpp_Runner.useUndefinedSanitizer": false,
101 "C_Cpp_Runner.useLeakSanitizer": false,
```

```
102 "C_Cpp_Runner.showCompilationTime": false,  
103 "C_Cpp_Runner.useLinkTimeOptimization": false,  
104 "C_Cpp_Runner.msvcSecureNoWarnings": false  
105 }
```

```
1  /*
2  ArrayBag interface for term project
3  CSCI 235 Spring 2024
4  */
5
6
7  #include "ArrayBag.hpp"
8
9  /** default constructor**/
10 template<class ItemType>
11 ArrayBag<ItemType>::ArrayBag(): item_count_(0)
12 {
13 } // end default constructor
14
15 /**
16  @return item_count_ : the current size of the bag
17  **/
18 template<class ItemType>
19 int ArrayBag<ItemType>::getCurrentSize() const
20 {
21     return item_count_;
22 } // end getCurrentSize
23
24 /**
25  @return true if item_count_ == 0, false otherwise
26  **/
27 template<class ItemType>
28 bool ArrayBag<ItemType>::isEmpty() const
29 {
30     return item_count_ == 0;
31 } // end isEmpty
32
33 /**
34  @return true if new_entry was successfully added to items_, false otherwise
35  **/
36 template<class ItemType>
37 bool ArrayBag<ItemType>::add(const ItemType& new_entry)
38 {
39     bool has_room = (item_count_ < DEFAULT_CAPACITY);
40     if (has_room)
41     {
42         items_[item_count_] = new_entry;
43         item_count_++;
44         return true;
45     } // end if
46
47     return false;
48 } // end add
49
```



```

50 /**
51  @return true if an_entry was successfully removed from items_, false otherwise
52  **/
53 template<class ItemType>
54 bool ArrayBag<ItemType>::remove(const ItemType& an_entry)
55 {
56     int found_index = getIndexOf(an_entry);
57     bool can_remove = !isEmpty() && (found_index > -1);
58     if (can_remove)
59     {
60         item_count--;
61         items_[found_index] = items_[item_count_];
62     } // end if
63
64     return can_remove;
65 } // end remove
66
67 /**
68  @post item_count_ == 0
69  **/
70 template<class ItemType>
71 void ArrayBag<ItemType>::clear()
72 {
73     item_count_ = 0;
74 } // end clear
75
76 /**
77  @return the number of times an_entry is found in items_
78  **/
79 template<class ItemType>
80 int ArrayBag<ItemType>::getFrequencyOf(const ItemType& an_entry) const
81 {
82     int frequency = 0;
83     int curr_index = 0;    // Current array index
84     while (curr_index < item_count_)
85     {
86         if (items_[curr_index] == an_entry)
87         {
88             frequency++;
89         } // end if
90
91         curr_index++;    // Increment to next entry
92     } // end while
93
94     return frequency;
95 } // end getFrequencyOf
96
97 /**
98  @return true if an_entry is found in items_, false otherwise
99  **/
100 template<class ItemType>
101 bool ArrayBag<ItemType>::contains(const ItemType& an_entry) const

```

```
102 {
103     return getIndexOf(an_entry) > -1;
104 } // end contains
105
106 // ***** PRIVATE METHODS *****//
107
108 /**
109     @param target to be found in items_
110     @return either the index target in the array items_ or -1,
111     if the array does not contain the target.
112 **/
113 template<class ItemType>
114 int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
115 {
116     bool found = false;
117     int result = -1;
118     int search_index = 0;
119     // If the bag is empty, item_count_ is zero, so loop is skipped
120     while (!found && (search_index < item_count_))
121     {
122
123         if (items_[search_index] == target)
124         {
125             found = true;
126             result = search_index;
127         }
128         else
129         {
130             search_index++;
131         } // end if
132     } // end while
133
134     return result;
135 } // end getIndexOf
136
137
```

```
1  /*
2  ArrayBag interface for term project
3  CSCI 235 Spring 2024
4  */
5
6  #ifndef ARRAY_BAG_
7  #define ARRAY_BAG_
8  #include <iostream>
9  #include <vector>
10
11 template <class ItemType>
12 class ArrayBag
13 {
14
15     public:
16     /** default constructor**/
17     ArrayBag();
18
19     /**
20      @return item_count_ : the current size of the bag
21      **/
22     int getCurrentSize() const;
23
24     /**
25      @return true if item_count_ == 0, false otherwise
26      **/
27     bool isEmpty() const;
28
29     /**
30      @return true if new_entry was successfully added to items_, false otherwise
31      **/
32     bool add(const ItemType &new_entry);
33
34     /**
35      @return true if an_entry was successfully removed from items_, false otherwise
36      **/
37     bool remove(const ItemType &an_entry);
38
39     /**
40      @post item_count_ == 0
41      **/
42     void clear();
43
44     /**
45      @return true if an_entry is found in items_, false otherwise
46      **/
47     bool contains(const ItemType &an_entry) const;
48
49     /**
```

```
50     @return the number of times an_entry is found in items_  
51     **/  
52     int getFrequencyOf(const ItemType &an_entry) const;  
53  
54     protected:  
55     static const int DEFAULT_CAPACITY = 100; //max size of items_ at 100 by default for this project  
56     ItemType items_[DEFAULT_CAPACITY];    // Array of bag items  
57     int item_count_;                      // Current count of bag items  
58  
59     /**  
60     @param target to be found in items_  
61     @return either the index target in the array items_ or -1,  
62     if the array does not contain the target.  
63     **/  
64     int getIndexOf(const ItemType &target) const;  
65  
66 }; // end ArrayBag  
67  
68 #include "ArrayBag.cpp"  
69 #endif  
70
```

```
1  /**
2   * @file Cavern.cpp
3   * @author Devin Chen
4   * @brief Cavern Class
5   * @date 3/6/2024
6   */
7  #include "Cavern.hpp"
8
9  /**
10   * Default constructor.
11   * Default-initializes all private members.
12   */
13  Cavern::Cavern(){
14      level_sum_ = 0;
15      tame_count_ = 0;
16  }
17  /**
18   * @param   : A reference to a Creature entering the Cavern
19   * @post    : If the given Creature is not already in the Cavern, add Creature to the Cavern and
20   *            updates the level sum and the tame Creature count if the creature is tame.
21   * @return  : returns true if a Creature was successfully added to the Cavern, false otherwise
22   *            : Hint: Use the above definition of equality will help determine if a Creature is already in the
23   *            Cavern
24   */
25  bool Cavern::enterCavern(const Creature &new_creature_added){
26      if(contains(new_creature_added)){
27          return false;
28      }
29      else{
30          add(new_creature_added);
31          level_sum_ += new_creature_added.getLevel();
32          if(new_creature_added.isTame()){
33              tame_count_++;
34          }
35          return true;
36      }
37  }
38  /**
39   * @param   : A reference to a Creature leaving the Cavern
40   * @return  : returns true if a creature was successfully removed from the Cavern (i.e. items_), false
41   *            otherwise
42   * @post    : removes the creature from the Cavern and updates the level sum.
43   *            * If the Creature is tame it also updates the tame count.
44   */
45  bool Cavern::exitCavern(const Creature &new_creature_leaving){
46      if(remove(new_creature_leaving)){
47          level_sum_ -= new_creature_leaving.getLevel();
48          if(new_creature_leaving.isTame()){
49              tame_count_--;
```

```

47     }
48     return true;
49 }
50 return false;
51 }
52 /**
53  * @return : The integer level count of all the creatures currently in the Cavern
54  **/
55 int Cavern::getLevelSum()const{
56     return level_sum_;
57 }
58 /**
59  * @return : The integer level count of all the creatures currently in the Cavern
60  **/
61 int Cavern::calculateAvgLevel()const{
62     if(getCurrentSize() == 0){
63         return 0;
64     }
65     return round(static_cast<double>(level_sum_) / getCurrentSize());
66 }
67 /**
68  * @return : The integer count of tame Creatures in the Cavern
69  **/
70 int Cavern::getTameCount()const{
71     return tame_count_;
72 }
73 /**
74  * @return : The percentage (double) of all the tame creatures in the Cavern
75  * @post : Computes the percentage of tame creatures in the Cavern rounded up to 2 decimal
76  places.
77  **/
78 double Cavern::calculateTamePercentage() const {
79     if (getCurrentSize() == 0) {
80         return 0;
81     }
82     double percentage = static_cast<double>(tame_count_) / static_cast<double>(getCurrentSize()) *
100;
83     return std::ceil(percentage * 100) / 100;
84 }
85 /**
86  * @param : A reference to a string representing a creature Category with value in ["UNKNOWN",
87  "UNDEAD", "MYSTICAL", "ALIEN"]
88  * @return : An integer tally of the number of creatures in the Cavern of the given category.
89  *If the argument string does not match one of the expected category values, the tally is zero.
90  *NOTE: no pre-processing of the input string necessary, only uppercase input will match.
91  **/
92 int Cavern::tallyCategory(const std::string &Category)const{
93     int tally = 0;
94     for(int i = 0; i < getCurrentSize(); i++){
95         if(items_[i].getCategory() == Category){
96             tally ++;
97         }
98     }
99 }

```

```

96 }
97 return tally;
98 }
99 /**
100 * @param : An integer representing the level threshold of the creatures to be removed from the
Cavern, with default value 0
101 * @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.
102 * @return : The number of creatures removed from the Cavern
103 */
104 int Cavern::releaseCreaturesBelowLevel(int threshold) {
105     int numremoved = 0;
106     if(threshold == 0){
107         numremoved = getCurrentSize();
108         clear();
109         level_sum_ = 0;
110         tame_count_ = 0;
111     }
112     if(threshold > 0){
113         for(int i = 0; i < getCurrentSize(); i++){
114             if(items_[i].getLevel() < threshold){
115                 exitCavern(items_[i]);
116                 numremoved ++;
117                 i--;
118             }
119         }
120     }
121 }
122 return numremoved;
123 }
124 /**
125 * @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
126 * @post : Removes all creatures from the Cavern whose category matches the given category. If no
category is given, removes all creatures from the Cavern.
127 * @return : The number of creatures removed from the Cavern
128 *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.
129 */
130 int Cavern::releaseCreaturesOfCategory(const std::string &value){
131     int numremoved = 0;
132     if(value == "ALL"){
133         for(int i = 0; i < getCurrentSize(); i++){
134             exitCavern(items_[i]);
135             numremoved++;
136         }
137     }
138     else{
139         for(int i = 0; i < getCurrentSize(); i++){
140             if(items_[i].getCategory() == value){
141                 exitCavern(items_[i]);
142                 numremoved++;

```

```

143     }
144 }
145 }
146 return numremoved;
147 }
148 /**
149  * @post   : Outputs a report of the creatures currently in the Cavern in the form:
150  * "UNKNOWN: [x]\nUNDEAD: [x]\nMYSTICAL: [x]\nALIEN: [x]\n\nAVERAGE LEVEL: [x]\nTAME:[x]%\n"
151  * Note that the average level should be rounded to the NEAREST integer, and the percentage of
152  * tame creatures in the Cavern should be rounded to 2 decimal places.
153
154  Example output:
155  UNKNOWN: 1
156  UNDEAD: 3
157  MYSTICAL: 2
158  ALIEN: 1
159
160  AVERAGE LEVEL: 5
161  TAME: 85.72%
162  */
163 void Cavern::cavernReport()const {
164     std::cout << "UNKNOWN: " << tallyCategory("UNKNOWN") << std::endl;
165     std::cout << "UNDEAD: " << tallyCategory("UNDEAD") << std::endl;
166     std::cout << "MYSTICAL: " << tallyCategory("MYSTICAL") << std::endl;
167     std::cout << "ALIEN: " << tallyCategory("ALIEN") << std::endl << std::endl;
168     std::cout << "AVERAGE LEVEL: " << calculateAvgLevel() << std::endl;
169     std::cout << "TAME: " << calculateTamePercentage() << "%" << std::endl;
170 }
171

```



```
1  /**
2   * @file Cavern.hpp
3   * @author Devin Chen
4   * @brief Cavern Class
5   * @date 3/6/2024
6   */
7  #include "ArrayBag.hpp"
8  #include "Creature.hpp"
9  #include <cmath>
10 #pragma once
11 class Cavern:public ArrayBag<Creature> {
12     public:
13     /**
14      * Default constructor.
15      * Default-initializes all private members.
16      */
17     Cavern();
18     /**
19      * @param   : A reference to a Creature entering the Cavern
20      * @post    : If the given Creature is not already in the Cavern, add Creature to the Cavern and
21                  updates the level sum and the tame Creature count if the creature is tame.
22      * @return  : returns true if a Creature was successfully added to the Cavern, false otherwise
23                  : Hint: Use the above definition of equality will help determine if a Creature is already in the
24                  Cavern
25      */
26     bool enterCavern(const Creature& creature);
27     /**
28      * @param   : A reference to a Creature leaving the Cavern
29      * @return  : returns true if a creature was successfully removed from the Cavern (i.e. items_), false
30                  otherwise
31      * @post   : removes the creature from the Cavern and updates the level sum.
32      * If the Creature is tame it also updates the tame count.
33      */
34     bool exitCavern(const Creature& creature);
35     /**
36      * @return  : The integer level count of all the creatures currently in the Cavern
37      */
38     int getLevelSum() const;
39     /**
40      * @return  : The integer level count of all the creatures currently in the Cavern
41      */
42     int calculateAvgLevel() const;
43     /**
44      * @return  : The integer count of tame Creatures in the Cavern
45      */
46     int getTameCount() const;
47     /**
48      * @return  : The percentage (double) of all the tame creatures in the Cavern
```

```

47  * @post   : Computes the percentage of tame creatures in the Cavern rounded up to 2 decimal
48             places.
49             **/
50             double calculateTamePercentage() const;
51             /**
52             * @param : A reference to a string representing a creature Category with value in ["UNKNOWN",
53             "UNDEAD", "MYSTICAL", "ALIEN"]
54             * @return : An integer tally of the number of creatures in the Cavern of the given category.
55             *If the argument string does not match one of the expected category values, the tally is zero.
56             *NOTE: no pre-processing of the input string necessary, only uppercase input will match.
57             **/
58             int tallyCategory(const std::string& category) const;
59             /**
60             * @param : An integer representing the level threshold of the creatures to be removed from the
61             Cavern, with default value 0
62             * @post  : Removes all creatures from the Cavern whose level is less than the given level. If no
63             level is given, removes all creatures from the Cavern. Ignore negative input.
64             * @return : The number of creatures removed from the Cavern
65             */
66             int releaseCreaturesBelowLevel(const int threshold = 0);
67             /**
68             * @param : A reference to a string representing a creature Category with a value in ["UNKNOWN",
69             "UNDEAD", "MYSTICAL", "ALIEN"], or default value "ALL" if no category is given
70             * @post  : Removes all creatures from the Cavern whose category matches the given category. If no
71             category is given, removes all creatures from the Cavern.
72             * @return : The number of creatures removed from the Cavern
73             *NOTE: no pre-processing of the input string necessary, only uppercase input will match. If the
74             input string does not match one of the expected category values, do not remove any creatures.
75             */
76             int releaseCreaturesOfCategory(const std::string& category = "ALL");
77             /**
78             * @post   : Outputs a report of the creatures currently in the Cavern in the form:
79             "UNKNOWN: [x]\nUNDEAD: [x]\nMYSTICAL: [x]\nALIEN: [x]\n\nAVERAGE LEVEL: [x]\nTAME:[x]%\n"
80             Note that the average level should be rounded to the NEAREST integer, and the percentage of
81             tame creatures in the Cavern should be rounded to 2 decimal places.
82             */
83             Example output:
84             UNKNOWN: 1
85             UNDEAD: 3
86             MYSTICAL: 2
87             ALIEN: 1
88             AVERAGE LEVEL: 5
89             TAME: 85.72%
90             */
91             void cavernReport() const;
92             private:
93             int level_sum_;
94             int tame_count_;
95             };

```

```
1  /*
2  CSCI235 Spring 2024
3  Project 1 - Creature Class
4  Georgina Woo
5  Nov 13 2023
6  Creature.cpp defines the constructors and private and public function implementation of the
   Creature class
7  */
8
9  #include "Creature.hpp"
10
11 /**
12   Default constructor.
13   Default-initializes all private members.
14   Default creature name: "NAMELESS".
15   Booleans are default-initialized to False.
16   Default enum value: UNKNOWN
17   Default Hitpoints and Level: 1.
18 */
19 Creature::Creature(): name_{"NAMELESS"}, category_{UNKNOWN}, hitpoints_{1}, level_{1},
   tame_{false}
20 {
21
22 }
23
24 /**
25   Parameterized constructor.
26   @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.
27   @param    : The category of the creature (a Category enum) with default value UNKNOWN
28   @param    : The creature's hitpoints (an integer) , with default value 1 if not provided, or if the
   value provided is 0 or negative
29   @param    : The creature's level (an integer), with default value 1 if not provided, or if the value
   provided is 0 or negative
30   @param    : A flag indicating whether the creature is tame, with default value False
31   @post     : The private members are set to the values of the corresponding parameters.
32   Hint: Notice the default arguments in the parameterized constructor.
33 */
34 Creature::Creature(const std::string& name, Category category, int hitpoints, int level, bool tame):
   category_{category}
35 {
36   if(!SetName(name))
37   {
38     name_ = "NAMELESS";
39   }
40
41   if(!SetHitpoints(hitpoints))
42   {
43     hitpoints_ = 1;
```

```

44     }
45     if(!setLevel(level))
46     {
47         level_ = 1;
48     }
49     tame_ = tame;
50
51 }
52
53 /**
54  @param : the name of the Creature, a reference to string
55  @post  : sets the Creature's name to the value of the parameter in UPPERCASE.
56           (convert any lowercase character to uppercase)
57           Only alphabetical characters are allowed.
58           : If the input contains non-alphabetic characters, do nothing.
59  @return : true if the name was set, false otherwise
60 */
61 bool Creature::setName(const std::string& name)
62 {
63     if (name.length() == 0)
64     {
65         return false;
66     }
67     else
68     {
69         std::string nameUpper = name;
70         for (int i = 0; i < name.length(); i++)
71         {
72             if (!isalpha(name[i]))
73             {
74                 return false;
75             }
76             else
77             {
78                 nameUpper[i] = toupper(name[i]);
79             }
80         }
81         name_ = nameUpper;
82         return true;
83     }
84 }
85
86 /**
87  @return : the name of the Creature
88 */
89 std::string Creature::getName() const
90 {
91     return name_;
92 }
93
94
95 /**

```

```

96     @param : the category of the Creature (an enum)
97     @post  : sets the Creature's category to the value of the parameter
98     */
99     void Creature::setCategory(const Category& category)
100    {
101        category_ = category;
102    }
103
104
105    /**
106        @return : the category of the Creature (in string form)
107    */
108    std::string Creature::getCategory() const
109    {
110        switch(category_)
111        {
112            case UNDEAD:
113                return "UNDEAD";
114            case MYSTICAL:
115                return "MYSTICAL";
116            case ALIEN:
117                return "ALIEN";
118            default:
119                return "UNKNOWN";
120        }
121    }
122
123    /**
124        @param : an integer that represents the creature's hitpoints
125        @pre   : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid
input)
126        @post  : sets the hitpoints private member to the value of the parameter
127        @return : true if the hitpoints were set, false otherwise
128    */
129    bool Creature::setHitpoints(const int& hitpoints)
130    {
131        if (hitpoints > 0)
132        {
133            hitpoints_ = hitpoints;
134            return true;
135        }
136        else
137        {
138            return false;
139        }
140    }
141
142
143    /**
144        @return : the value stored in hitpoints_
145    */
146    int Creature::getHitpoints() const

```

```

147 {
148     return hitpoints_;
149 }
150
151 /**
152     @param : an integer level
153     @pre   : level > 0 : Characters cannot have 0 or negative levels (do nothing for invalid input)
154     @post  : sets the level private member to the value of the parameter
155     @return : true if the level was set, false otherwise
156 */
157 bool Creature::setLevel(const int& level)
158 {
159     if (level > 0)
160     {
161         level_ = level;
162         return true;
163     }
164     else
165     {
166         return false;
167     }
168 }
169
170
171 /**
172     @return : the value stored in level_
173 */
174 int Creature::getLevel() const
175 {
176     return level_;
177 }
178
179
180 /**
181     @param : a boolean value
182     @post  : sets the tame flag to the value of the parameter
183 */
184 void Creature::setTame(const bool& tame)
185 {
186     tame_ = tame;
187 }
188
189
190 /**
191     @return true if the creature is tame, false otherwise
192     Note: this is an accessor function and must follow the same convention as all accessor functions
193     even if it is not called getTame
194 */
195 bool Creature::isTame() const
196 {
197     return tame_;
198 }

```

```

198
199 /**
200     @post    : displays Creature data in the form:
201     "[NAME]\n
202     Category: [CATEGORY]\n
203     Level: [LEVEL]\n
204     Hitpoints: [Hitpoints]\n
205     Tame: [TRUE/FALSE]"
206 */
207 void Creature::display() const
208 {
209     std::cout << name_ << std::endl;
210     std::cout << "Category: " << getCategory() << std::endl;
211     std::cout << "Level: " << level_ << std::endl;
212     std::cout << "Hitpoints: " << hitpoints_ << std::endl;
213     std::cout << "Tame: " << (tame_ ? "TRUE" : "FALSE") << std::endl;
214 }
215
216 /**
217  * @param    : a const reference to the right hand side of the == operator.
218  * @return    : Returns true if the right hand side creature is "equal", false otherwise.
219  * Two creatures are equal if they have the same name, same category, same level, and if they're
220  * either both tame or both not
221  * NOTE: By this definition, only the aforementioned subset of the creature's attributes must be
222  * equal for two creatures to be deemed "equal".
223
224  * Example: In order for creature1 to be == to creature2 we only need:
225  * The same name
226  * The same category
227  * The same level
228  * They must either be both tame or both not
229 */
230 bool Creature::operator==(const Creature &reference) const{
231     if(reference.name_ == name_ && reference.level_ == level_ && reference.category_ == category_
232     && reference.tame_ == tame_){
233         return true;
234     }
235     return false;
236 }
237
238 /**
239  * @param    : a const reference to the right hand side of the != operator.
240  * @return    : Returns true if the right hand side creature is NOT "equal" (!=), false
241  * otherwise. Two creatures are NOT equal if any of their name, category or level are not equal, or if
242  * one is tame and the other is not.
243  * NOTE: By this definition, one or more of the aforementioned subset of the creature's attributes
244  * only must be different for two creatures to be deemed "NOT equal".
245 */
246 bool Creature::operator!=(const Creature &reference) const{
247     if(reference.name_ == name_ && reference.level_ == level_ && reference.category_ == category_
248     && reference.tame_ == tame_){
249         return false;

```

```
244     }  
245     return true;  
246 }
```



```
1  /*
2  CSCI235 Spring 2024
3  Project 1 - Creature Class
4  Georgina Woo
5  Nov 13 2023
6  Creature.hpp declares the Creature class along with its private and public members
7  */
8  #ifndef CREATURE_HPP_
9  #define CREATURE_HPP_
10 #include <iostream>
11 #include <string>
12 #include <cctype>
13
14
15 class Creature
16 {
17     public:
18         enum Category {UNKNOWN, UNDEAD, MYSTICAL, ALIEN};
19         /**
20          Default constructor.
21          Default-initializes all private members.
22          Default creature name: "NAMELESS".
23          Booleans are default-initialized to False.
24          Default enum value: UNKNOWN
25          Default Hitpoints and Level: 1.
26         */
27         Creature();
28
29         /**
30          Parameterized constructor.
31          @param    : The name of the creature (a string)
32          @param    : The category of the creature (a Category enum) with default value UNKNOWN
33          @param    : The creature's hitpoints (an integer), with default value 1 if not provided, or if
the value provided is 0 or negative
34          @param    : The creature's level (an integer), with default value 1 if not provided, or if the
value provided is 0 or negative
35          @param    : A flag indicating whether the creature is tame, with default value False
36          @post     : The private members are set to the values of the corresponding parameters.
37          Hint: Notice the default arguments in the parameterized constructor.
38         */
39         Creature(const std::string& name, Category category = UNKNOWN, int hitpoints = 1, int level =
1, bool tame = false);
40
41         /**
42          @param : the name of the Creature, a string
43          @post  : sets the Creature's name to the value of the parameter in UPPERCASE (convert any
lowercase character to upppercase
44                  Only alphabetical characters are allowed.
45                  : If the input contains non-alphabetic characters, do nothing.
```

```

46         @return : true if the name was set, false otherwise
47     */
48     bool setName(const std::string& name);
49
50     /**
51         @return : the name of the Creature
52     */
53     std::string getName() const;
54
55
56     /**
57         @param : the category of the Creature (an enum)
58         @post  : sets the Creature's category to the value of the parameter
59     */
60     void setCategory(const Category& category);
61
62
63     /**
64         @return : the race of the Creature (in string form)
65     */
66     std::string getCategory() const;
67
68     /**
69         @param : an integer that represents the creature's hitpoints
70         @pre   : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid
input)
71         @post  : sets the hitpoints private member to the value of the parameter
72         @return : true if the hitpoints were set, false otherwise
73     */
74     bool setHitpoints(const int& hitpoints);
75
76
77     /**
78         @return : the value stored in hitpoints_
79     */
80     int getHitpoints() const;
81
82     /**
83         @param : an integer level
84         @pre   : level > 0 : Creatures cannot have 0 or negative levels (do nothing for invalid input)
85         @post  : sets the level private member to the value of the parameter
86         @return : true if the level was set, false otherwise
87     */
88     bool setLevel(const int& level);
89
90
91     /**
92         @return : the value stored in level_
93     */
94     int getLevel() const;
95
96

```

```

97     /**
98         @param : a boolean value
99         @post  : sets the tame flag to the value of the parameter
100    */
101    void setTame(const bool& tame);
102
103
104    /**
105        @return true if the Creature is tame, false otherwise
106        Note: this is an accessor function and must follow the same convention as all accessor
functions even if it is not called getTame
107    */
108    bool isTame() const;
109
110    /**
111        @post   : displays Creature data in the form:
112        "[NAME]\n
113        Category: [CATEGORY]\n
114        Level: [LEVEL]\n
115        Hitpoints: [Hitpoints]\n
116        Tame: [TRUE/FALSE]"
117    */
118    void display() const;
119
120    /**
121        * @param    : a const reference to the right hand side of the == operator.
122        * @return    : Returns true if the right hand side creature is "equal", false otherwise.
123        * Two creatures are equal if they have the same name, same category, same level, and if
they're either both tame or both not
124        * NOTE: By this definition, only the aforementioned subset of the creature's attributes must be
equal for two creatures to be deemed "equal".
125
126        * Example: In order for creature1 to be == to creature2 we only need:
127        * The same name
128        * The same category
129        * The same level
130        * They must either be both tame or both not
131    */
132    bool operator==(const Creature &reference) const;
133
134    /**
135        * @param    : a const reference to the right hand side of the != operator.
136        * @return    : Returns true if the right hand side creature is NOT "equal" (!=), false
137        * 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.
138        * 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".
139    */
140    bool operator!=(const Creature &reference) const;
141
142    private:
143        // The name of the creature (a string in UPPERCASE)

```

```

144     std::string name_;
145     // The category of the creature (an enum)
146     Category category_;
147     // The creature's hitpoints (a non-zero, non-negative integer)
148     int hitpoints_;
149     // The creature's level (a non-zero, non-negative integer)
150     int level_;
151     // A flag indicating whether the creature is tame
152     bool tame_;
153
154 };
155
156 #endif

```

#### ▼ Makefile

 Download

```

1  CXX = g++
2  CXXFLAGS = -std=c++17 -g -Wall -O2
3
4  PROG ?= main
5  OBJS = Creature.o main.o Cavern.o
6
7  all: $(PROG)
8
9  .cpp.o:
10     $(CXX) $(CXXFLAGS) -c -o $@ $<
11
12  $(PROG): $(OBJS)
13     $(CXX) $(CXXFLAGS) -o $@ $(OBJS)
14
15  clean:
16     rm -rf $(EXEC) *.o *.out main
17
18  rebuild: clean all
19

```

#### ▼ README.md

 Download

```

1  [![Review Assignment Due Date](https://classroom.github.com/assets/deadline-readme-button-
2  24ddc0f5d75046c5622901739e7c5dd533143b0c8e959d652212380cedb1ea36.svg)]
3  (https://classroom.github.com/a/tgc2wzn0)
4  # Project3
5
6  The project specification can be found on Blackboard

```