Slimes and Aztec Gold

Table of Contents

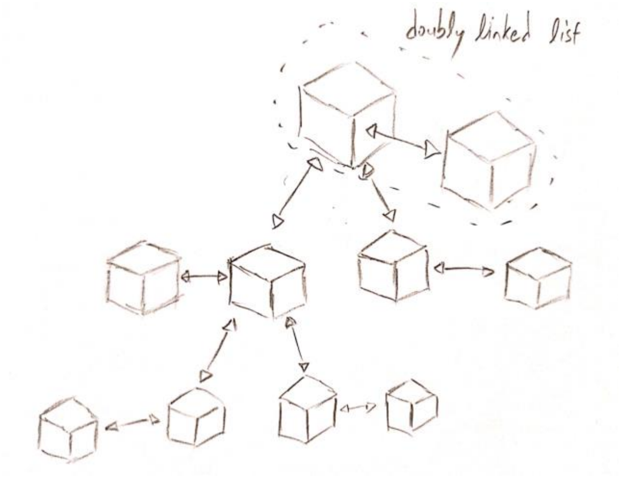
Contents

Link to Video Demonstration

https://youtu.be/JMnUIj8UCHs

Introduction and Description of Software Prototype Application

This game is about an adventure of protagonist slime and his friends in search of Aztec Gold. At the start, the player can choose one of the slimes as his main character and receive special starting boost to the slime. Then, he can change the name of the selected slime. The other two slimes will accompany him as party members. After finished all the setup, the party will be deployed automatically into the starting room. There will be a battle with monsters in the room and the party can take different action after defeated the monster. The player can check surrounding to find Aztec Gold or precious gacha key, use items, boost attributes by consuming gacha key with gacha machine, or move to other rooms. The game ends when the Aztec Gold is found, or all slimes are dead.



This is the illustration of the world. Each cube represents a room.

Inheritance and Derived Classes

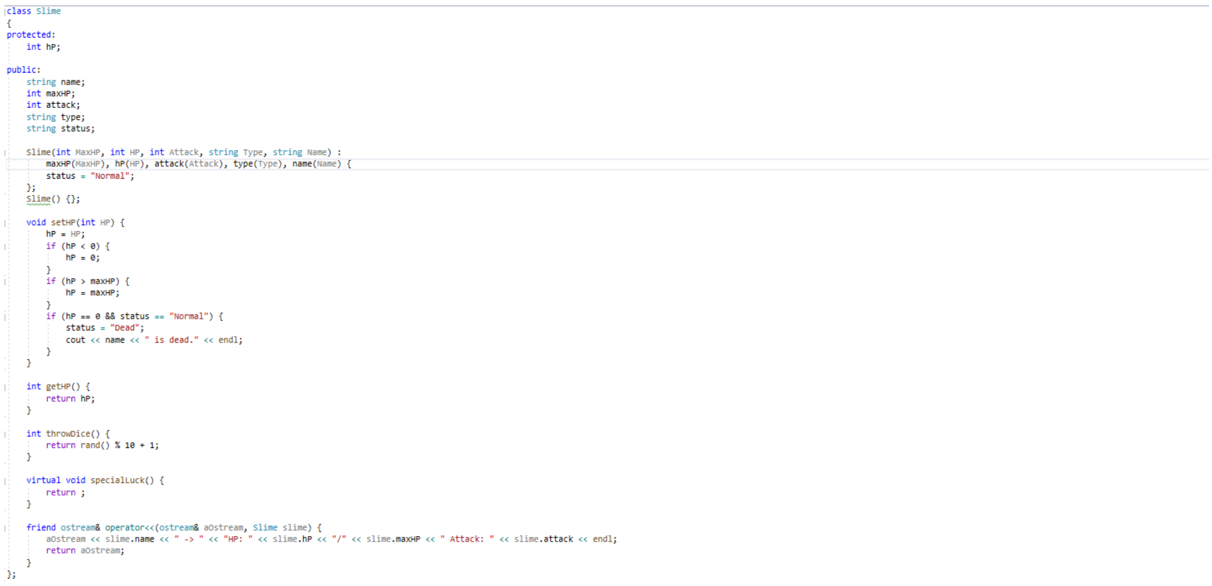
Task Description

In this game, the player can control a party of three slimes with different species. However, the slimes have different element attribute and different skills, but with a lot of common attributes too. At the start of the game, the player can select one of the slimes as his main character. Thus, a mechanism that could reduce the workload of creating different classes of slime is needed.

Concept

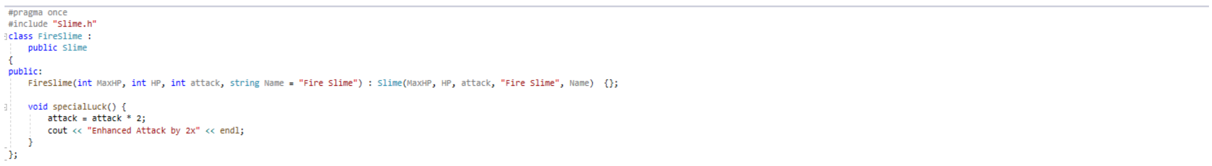
In C++, inheritance and derived classes is one of the ways to achieve the task. First, I create a class that has all the common attribute of different species of slimes. Then, I create three specific classes of slime and set these classes as derived classes of the base class. Automatically, these three specific classes will inherit all the attributes and functions mark as protected or public from the base class. Thus, gone the need to implement the common attributes over and over. Public, protected or private specifier are data hiding features of object-oriented programming language to accomplish the “encapsulation” concept of OOP’s four pillars. I had used getter and setter for protected and private attributes too. Setter allow validation of input and getter allow modification to the attribute without changes to client code.

Implementation & Output



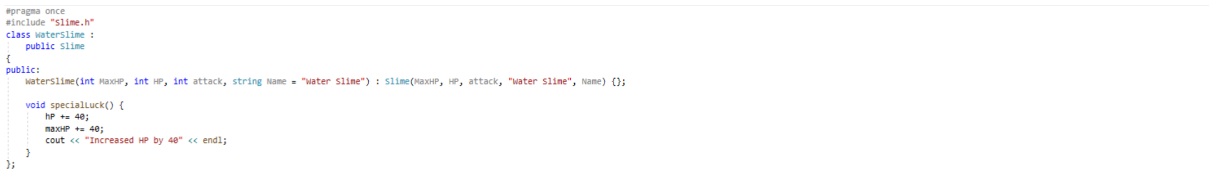
```cpp
class Slime
{
protected:
    int hP;

public:
    string name;
    int maxHP;
    int attack;
    string type;
    string status;

    Slime(int MaxHP, int HP, int Attack, string Type, string Name) :
        maxHP(MaxHP), hP(HP), attack(Attack), type(Type), name(Name) {
        status = "Normal";
    };
    Slime() {};

    void setHP(int HP) {
        hP = HP;
        if (hP < 0) {
            hP = 0;
        }
        if (hP > maxHP) {
            hP = maxHP;
        }
        if (hP == 0 && status == "Normal") {
            status = "Dead";
            cout << name << " is dead." << endl;
        }
    }

    int getHP() {
        return hP;
    }

    int throwDice() {
        return rand() % 10 + 1;
    }

    virtual void specialLuck() {
        return ;
    }

    friend ostream& operator<<(ostream& aOstream, Slime slime) {
        aOstream << slime.name << " -> " << "HP: " << slime.hP << "/" << slime.maxHP << " Attack: " << slime.attack << endl;
        return aOstream;
    }
};
```

Figure: base class Slime (Slime.h)



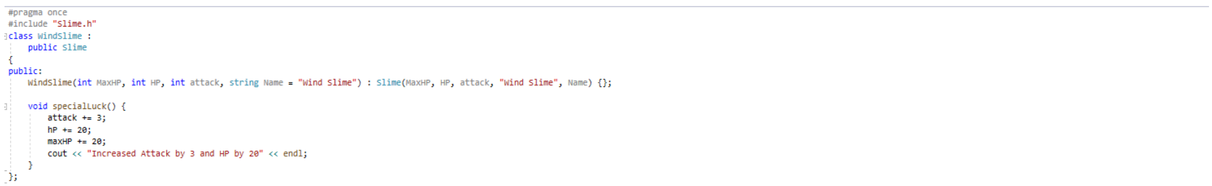
```cpp
#pragma once
#include "Slime.h"
class FireSlime :
    public Slime
{
public:
    FireSlime(int MaxHP, int HP, int attack, string Name = "Fire Slime") : Slime(MaxHP, HP, attack, "Fire Slime", Name)  {};

    void specialLuck() {
        attack = attack * 2;
        cout << "Enhanced Attack by 2x" << endl;
    }
};
```

Figure: derived class FireSlime (FireSlime.h)



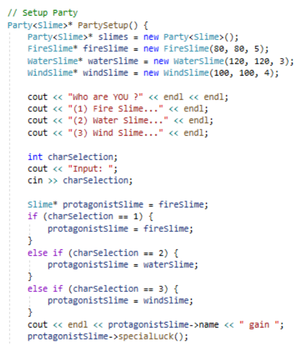
```cpp
#pragma once
#include "Slime.h"
class WaterSlime :
    public Slime
{
public:
    WaterSlime(int MaxHP, int HP, int attack, string Name = "Water Slime") : Slime(MaxHP, HP, attack, "Water Slime", Name) {};

    void specialLuck() {
        hP += 40;
        maxHP += 40;
        cout << "Increased HP by 40" << endl;
    }
};
```

Figure: derived class WaterSlime (WaterSlime.h)



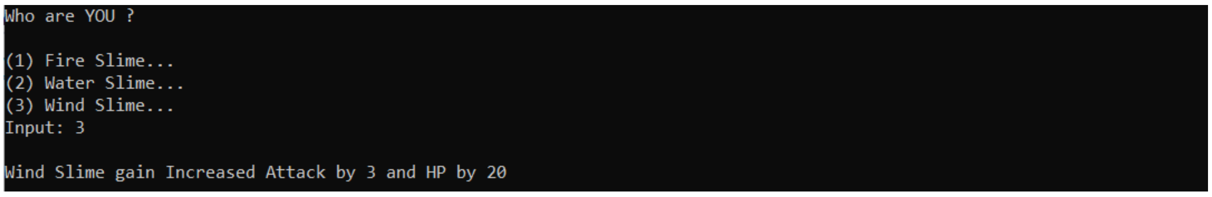
```cpp
#pragma once
#include "Slime.h"
class WindSlime :
    public Slime
{
public:
    WindSlime(int MaxHP, int HP, int attack, string Name = "Wind Slime") : Slime(MaxHP, HP, attack, "Wind Slime", Name) {};

    void specialLuck() {
        attack += 3;
        hP += 20;
        maxHP += 20;
        cout << "Increased Attack by 3 and HP by 20" << endl;
    }
};
```

Figure: derived class WindSlime (WindSlime.h)



```cpp
// Setup Party
Party<Slime>* PartySetup() {
    Party<Slime>* slimes = new Party<Slime>();
    FireSlime* fireSlime = new FireSlime(80, 80, 5);
    WaterSlime* waterSlime = new WaterSlime(120, 120, 3);
    WindSlime* windSlime = new WindSlime(100, 100, 4);

    cout << "Who are YOU ?" << endl << endl;
    cout << "(1) Fire Slime..." << endl;
    cout << "(2) Water Slime..." << endl;
    cout << "(3) Wind Slime..." << endl;

    int charSelection;
    cout << "Input: ";
    cin >> charSelection;

    Slime* protagonistSlime = fireSlime;
    if (charSelection == 1) {
        protagonistSlime = fireSlime;
    }
    else if (charSelection == 2) {
        protagonistSlime = waterSlime;
    }
    else if (charSelection == 3) {
        protagonistSlime = windSlime;
    }
    cout << endl << protagonistSlime->name << " gain ";
    protagonistSlime->specialLuck();
```

Figure: printing name of derived class with the use of pointer to base class (CodeReuse.h)



```
Who are YOU ?

(1) Fire Slime...
(2) Water Slime...
(3) Wind Slime...
Input: 3

Wind Slime gain Increased Attack by 3 and HP by 20
```

Figure: console output

Troubleshooting

To come up with the idea of how the game should be played is one of the hardest tasks. The design of models is not a hard job after I had the big picture and the syntax for inheritance is the easiest.

Friend Operator Overloading

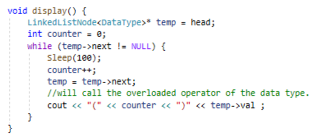
Task Description

In the program, it will print out the status of party members at different intervals. However, the party member is an object of slime, which is a composite data type, but not a primitive data type. For a console application, we will need to use << operator to print to standard output, but only primitive data type can be understood natively. To print composite data type to standard output, we will need to get the attributes of the object and print it one by one. Moreover, I use template class for the implementation of the linked list and thus composite data type must handle the printing by itself. Luckily, C++ does provide operator overloading for this task.

Concept

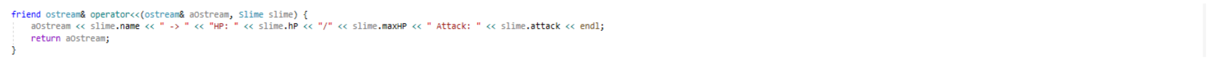
Operator overloading provides the programmer to implement different behaviours for different objects when the object interacts with the operator. With the friend specifier, the function is defined in the class but is not a member function of the class. However, a friend function has all the right to access all private and protected members of the class. Thus, friend specifier often uses together with operator overloading as operator is not member function of a class. In my implementation, an object of slime will call overloaded operator function when interacting with << operator and print out the necessary information.

Implementation & Output



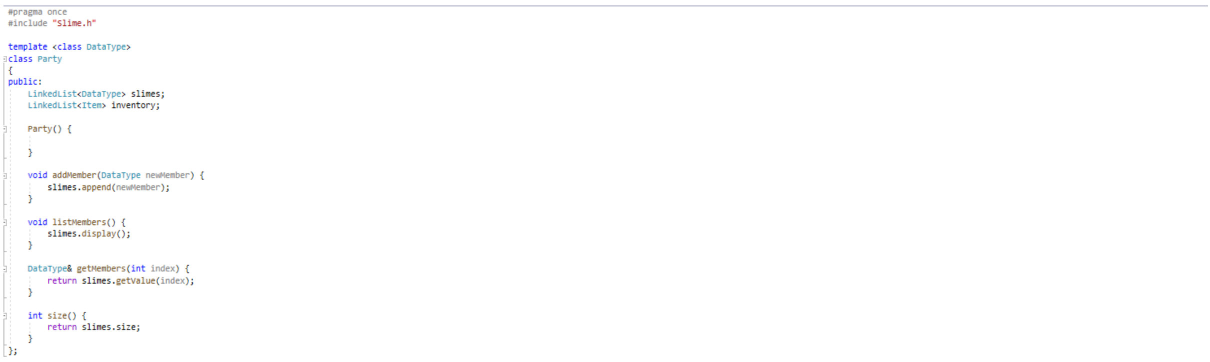
```cpp
void display() {
    LinkedListNode<DataType>* temp = head;
    int counter = 0;
    while (temp->next != NULL) {
        Sleep(100);
        counter++;
        temp = temp->next;
        //will call the overloaded operator of the data type.
        cout << "(" << counter << ")" << temp->val ;
    }
}
```

Figure: display the object in the linked list, but the DataType is generic class, thus friend operator overloading is needed (LinkedList.h)



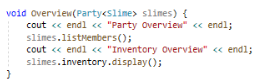
```cpp
friend ostream& operator<<(ostream& aOstream, Slime slime) {
    aOstream << slime.name << " -> " << "HP: " << slime.hP << "/" << slime.maxHP << " Attack: " << slime.attack << endl;
    return aOstream;
}
```

Figure: overloaded operator of Slime class (Slime.h)



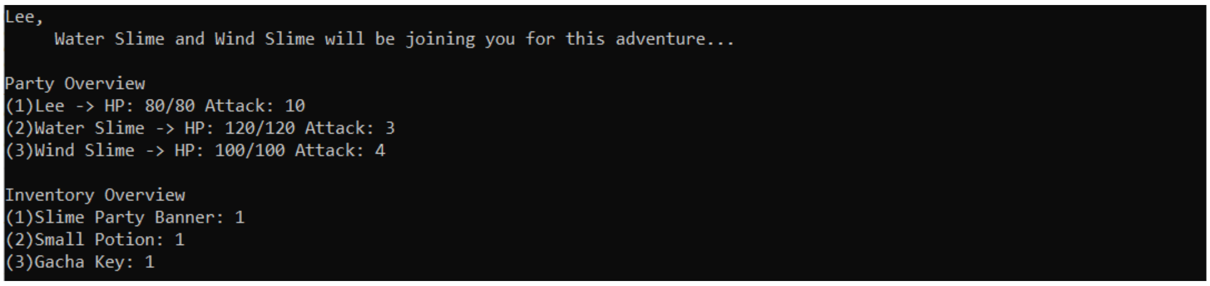
```cpp
#pragma once
#include "Slime.h"

template <class DataType>
class Party
{
public:
    LinkedList<DataType> slimes;
    LinkedList<Item> inventory;

    Party() {

    }

    void addMember(DataType newMember) {
        slimes.append(newMember);
    }

    void listMembers() {
        slimes.display();
    }

    DataType& getMembers(int index) {
        return slimes.getValue(index);
    }

    int size() {
        return slimes.size;
    }
};
```

Figure: party class that hold a singly linked list of slimes (Party.h)



```cpp
void Overview(Party<Slime> slimes) {
    cout << endl << "Party Overview" << endl;
    slimes.listMembers();
    cout << endl << "Inventory Overview" << endl;
    slimes.inventory.display();
}
```

Figure: print out the slimes with overloaded operator (CodeReuse.h)



```
Lee,
     Water Slime and Wind Slime will be joining you for this adventure...

Party Overview
(1)Lee -> HP: 80/80 Attack: 10
(2)Water Slime -> HP: 120/120 Attack: 3
(3)Wind Slime -> HP: 100/100 Attack: 4

Inventory Overview
(1)Slime Party Banner: 1
(2)Small Potion: 1
(3)Gacha Key: 1
```

Figure: console output

Troubleshooting

The challenging part is to realise the necessity to overload << operator when using a linked list implemented with template class. Otherwise, overload << operator is a common task and I had done it in the problem set 1.

Polymorphism

Task Description

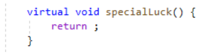
In this game, the player can control a party of three slimes, but only one slime will be selected as the main character and receive special starting abilities boost. At the start of the game, the player can select one of the slimes as his main character and the selected slime will need to invoke its hidden ability. In procedural programming, this is a complicated task as the client code will need to call the same function of different slime but have different behaviour.

Concept

Polymorphism is the ability of an object to take on many forms and as one of the four pillars of object-oriented programming, come to the save. In C++, the implementation of polymorphism is through inheritance/interface and virtual function. In my implementation, a template function of speciaLuck() that provide starting boost is implemented in the base class, and each derived class will override the function with its implementation. If the created object is fire slime and it is pointed by a pointer with a data type of pointer to base class Slime, fire slime's specialLuck() function will be invoked even the pointer has a data type of pointer to base class Slime.

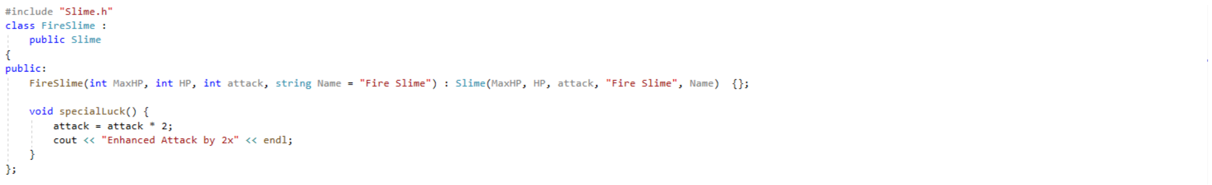
With polymorphism, we can declare a pointer to Slime named as protagonistSlime and working on the different selected slime only through this pointer without the need of writing many if else statement all over the program. At the later stage of program execution, there is no need to check which slime is the protagonist but could just refer to the pointer.

Implementation & Output



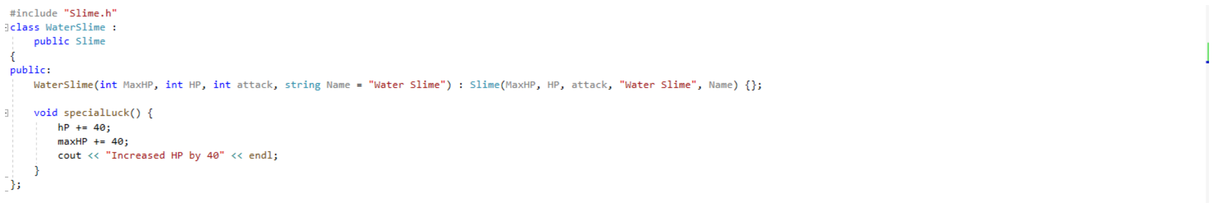
```cpp
    virtual void specialLuck() {
        return ;
    }
```

<u>Figure: template function of specialLuck() with virtual specifier in base class Slime (Slime.h)</u>



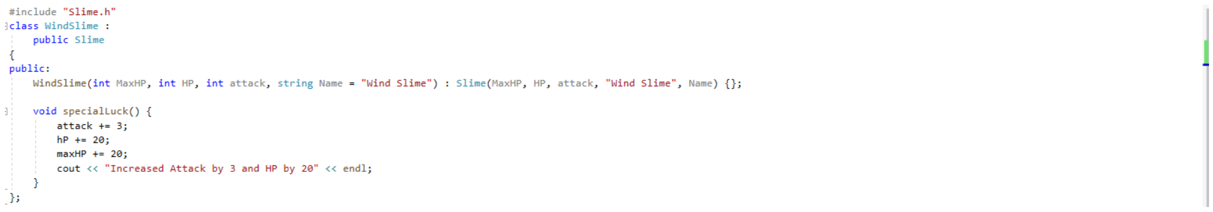
```cpp
#include "Slime.h"
class FireSlime :
    public Slime
{
public:
    FireSlime(int MaxHP, int HP, int attack, string Name = "Fire Slime") : Slime(MaxHP, HP, attack, "Fire Slime", Name)  {};

    void specialLuck() {
        attack = attack * 2;
        cout << "Enhanced Attack by 2x" << endl;
    }
};
```

<u>Figure: override specialLuck() function in derived class Fire Slime (FireSlime.h)</u>



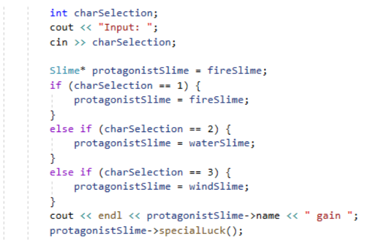
```cpp
#include "Slime.h"
class WaterSlime :
    public Slime
{
public:
    WaterSlime(int MaxHP, int HP, int attack, string Name = "Water Slime") : Slime(MaxHP, HP, attack, "Water Slime", Name) {};

    void specialLuck() {
        hP += 40;
        maxHP += 40;
        cout << "Increased HP by 40" << endl;
    }
};
```

<u>Figure: override specialLuck() function in derived class Water Slime (WaterSlime.h)</u>



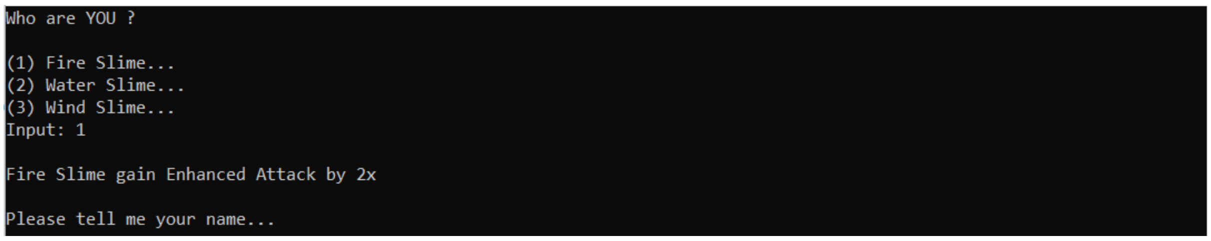
```cpp
#include "Slime.h"
class WindSlime :
    public Slime
{
public:
    WindSlime(int MaxHP, int HP, int attack, string Name = "Wind Slime") : Slime(MaxHP, HP, attack, "Wind Slime", Name) {};

    void specialLuck() {
        attack += 3;
        hP += 20;
        maxHP += 20;
        cout << "Increased Attack by 3 and HP by 20" << endl;
    }
};
```

<u>Figure: override specialLuck() function in derived class Wind Slime (WindSlime.h)</u>

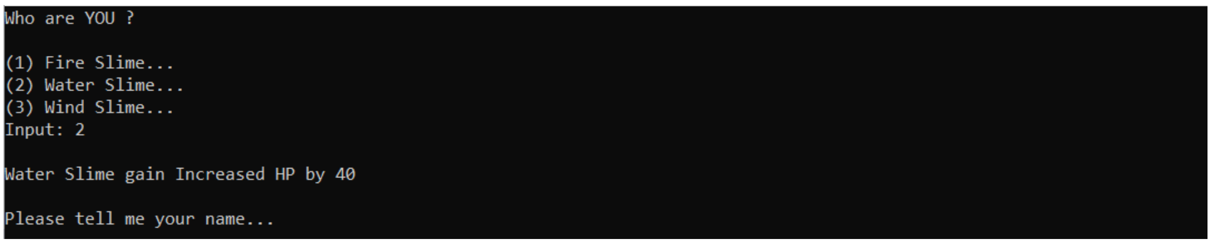


```cpp
        int charSelection;
        cout << "Input: ";
        cin >> charSelection;

        Slime* protagonistSlime = fireSlime;
        if (charSelection == 1) {
            protagonistSlime = fireSlime;
        }
        else if (charSelection == 2) {
            protagonistSlime = waterSlime;
        }
        else if (charSelection == 3) {
            protagonistSlime = windSlime;
        }
        cout << endl << protagonistSlime->name << " gain ";
        protagonistSlime->specialLuck();
```

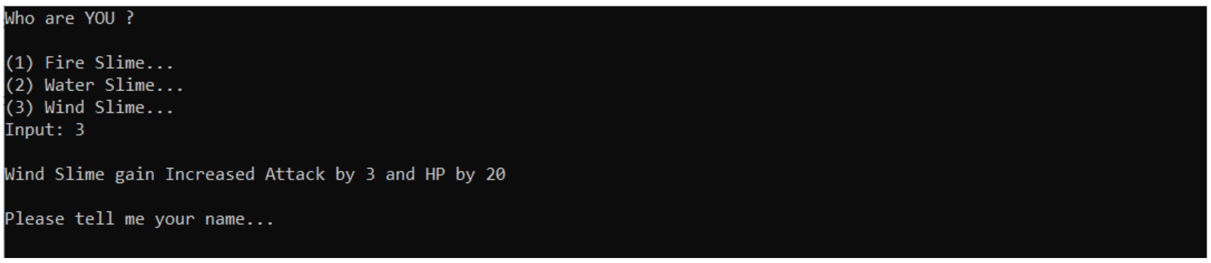
<u>Figure: assign different slime to the protagonistSlime pointer based on user selection and execute the specialLuck() function which will have different behaviour based on selected slime. Polymorphism, an object (protagonistSlime pointer) can have different forms (different enhancement on the selected slime). (CodeReuse.h)</u>



```
Who are YOU ?

(1) Fire Slime...
(2) Water Slime...
(3) Wind Slime...
Input: 1

Fire Slime gain Enhanced Attack by 2x

Please tell me your name...
```



```
Who are YOU ?

(1) Fire Slime...
(2) Water Slime...
(3) Wind Slime...
Input: 2

Water Slime gain Increased HP by 40

Please tell me your name...
```



```
Who are YOU ?

(1) Fire Slime...
(2) Water Slime...
(3) Wind Slime...
Input: 3

Wind Slime gain Increased Attack by 3 and HP by 20

Please tell me your name...
```

Figure: console output

Troubleshooting

As a beginner in C++, this task does have some tricky sides. I had fallen into it as I did not notice that C++ will carry out object slicing when an object of a derived class is assigned to an instance of base class and part of the derived class's specific information is lost. Thus, the virtual function of the derived class will not be invoked as there is no information about it. The solution is to use a pointer to base class to refer to the derived class.

Array

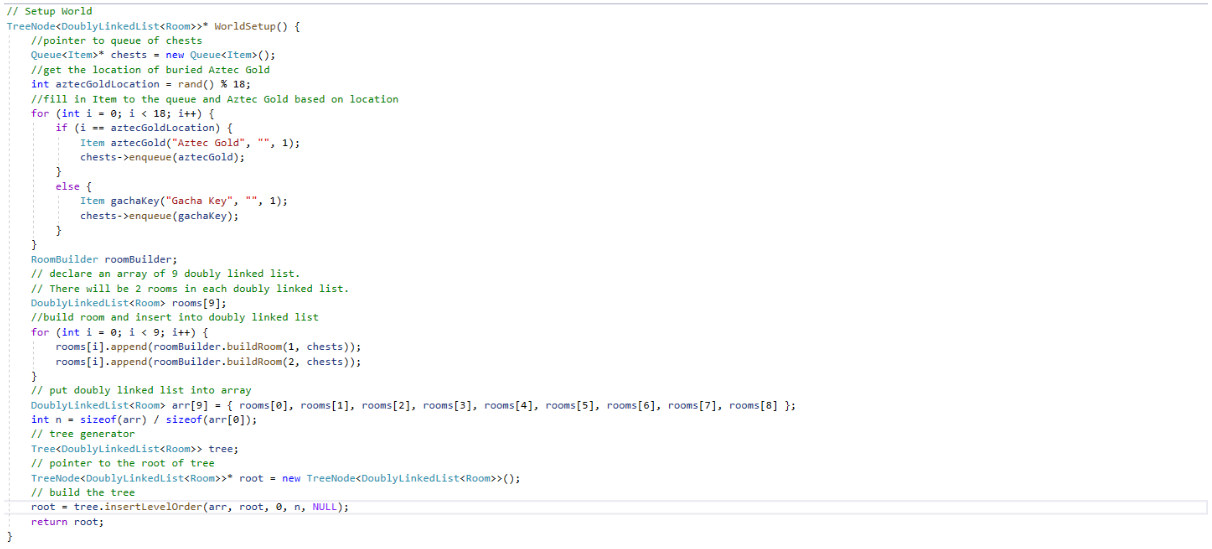
Task Description

I need to generate a world that consists of many connected rooms while in each room there are four doors to move into four different directions (Front, Back, Left, Right). Thus, I discovered a binary tree could be used, child nodes for left, right direction and parent node for back direction. The generation of tree and connection between rooms needs to be done automatically from existing rooms to reduce human error. Therefore, I decided to use an array to store these rooms and generate the tree from the array.

Concept

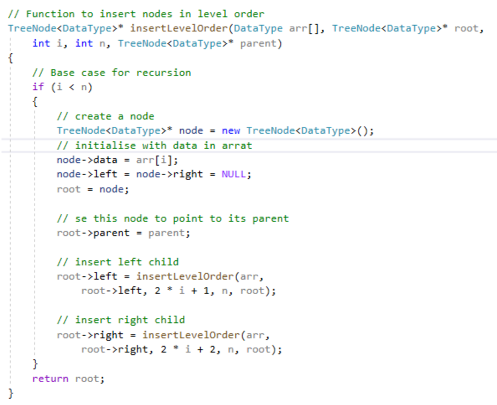
The using of array instead of linked list is preferred in this case due to random access is very frequent while generating the tree. Array is a type of data structure that store data in sequence, thus the time complexity to fetch an element is O(1) as the address of element could be acquired by pointer arithmetic, but linked list does require traverse all element in the worst-case scenario, leading to O(n). Moreover, page faults could happen if the pointer in linked list did not point to memory in RAM, lead to performance hits. In my implementation, I do know the number of rooms required and could allocate the correct amount of memory for the array without the risk of wasting unused space. Furthermore, array does use less memory than linked lists without the need of at least one pointer in each node.

Implementation & Output



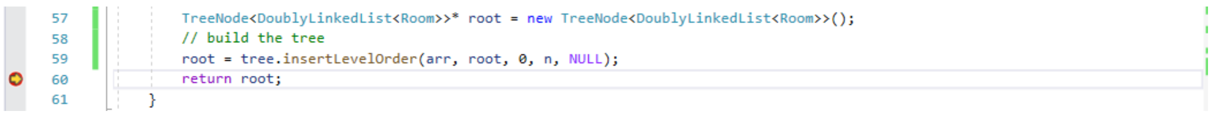
```
// Setup World
TreeNode<DoublyLinkedList<Room>>* WorldSetup() {
    //pointer to queue of chests
    Queue<Item>* chests = new Queue<Item>();
    //get the location of buried Aztec Gold
    int aztecGoldLocation = rand() % 18;
    //fill in Item to the queue and Aztec Gold based on location
    for (int i = 0; i < 18; i++) {
        if (i == aztecGoldLocation) {
            Item aztecGold("Aztec Gold", "", 1);
            chests->enqueue(aztecGold);
        }
        else {
            Item gachaKey("Gacha Key", "", 1);
            chests->enqueue(gachaKey);
        }
    }
    RoomBuilder roomBuilder;
    // declare an array of 9 doubly linked list.
    // There will be 2 rooms in each doubly linked list.
    DoublyLinkedList<Room> rooms[9];
    //build room and insert into doubly linked list
    for (int i = 0; i < 9; i++) {
        rooms[i].append(roomBuilder.buildRoom(1, chests));
        rooms[i].append(roomBuilder.buildRoom(2, chests));
    }
    // put doubly linked list into array
    DoublyLinkedList<Room> arr[9] = { rooms[0], rooms[1], rooms[2], rooms[3], rooms[4], rooms[5], rooms[6], rooms[7], rooms[8] };
    int n = sizeof(arr) / sizeof(arr[0]);
    // tree generator
    Tree<DoublyLinkedList<Room>> tree;
    // pointer to the root of tree
    TreeNode<DoublyLinkedList<Room>>* root = new TreeNode<DoublyLinkedList<Room>>();
    // build the tree
    root = tree.insertLevelOrder(arr, root, 0, n, NULL);
    return root;
}
```

Figure: pass the array into tree.insertLevelOrder() to generate tree from array (CodeReuse.h)



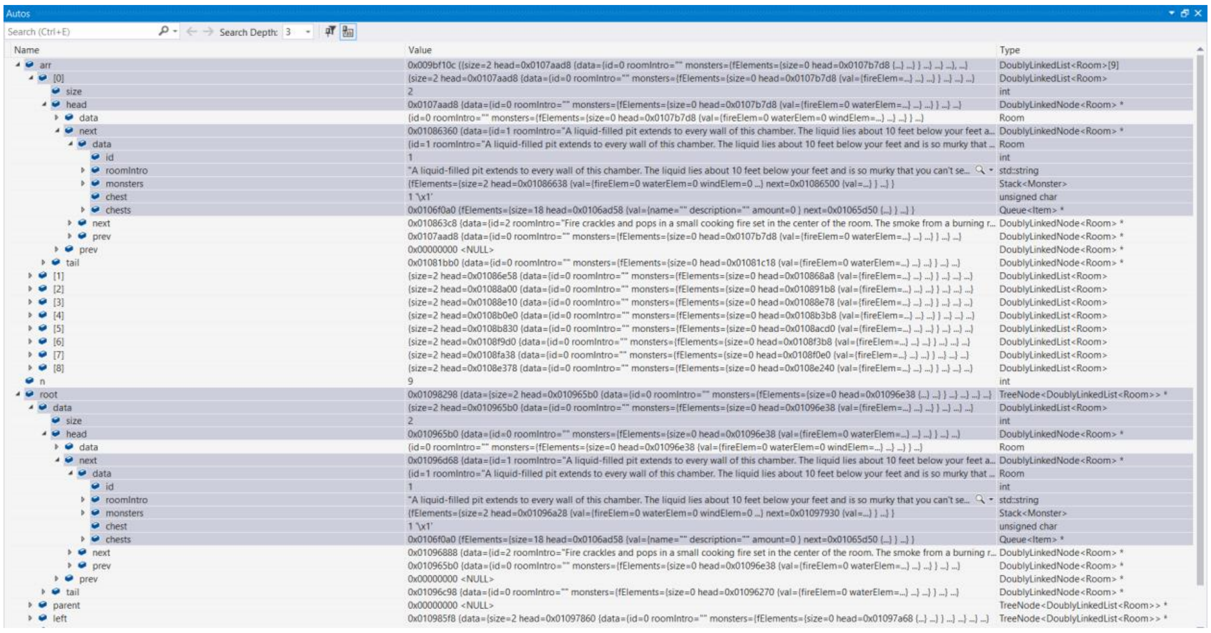
```
// Function to insert nodes in level order
TreeNode<DataType>* insertLevelOrder(DataType arr[], TreeNode<DataType>* root,
    int i, int n, TreeNode<DataType>* parent)
{
    // Base case for recursion
    if (i < n)
    {
        // create a node
        TreeNode<DataType>* node = new TreeNode<DataType>();
        // initialise with data in arrat
        node->data = arr[i];
        node->left = node->right = NULL;
        root = node;

        // se this node to point to its parent
        root->parent = parent;

        // insert left child
        root->left = insertLevelOrder(arr,
            root->left, 2 * i + 1, n, root);

        // insert right child
        root->right = insertLevelOrder(arr,
            root->right, 2 * i + 2, n, root);
    }
    return root;
}
```

Figure: random access the array and generate node from the element (Tree.h)



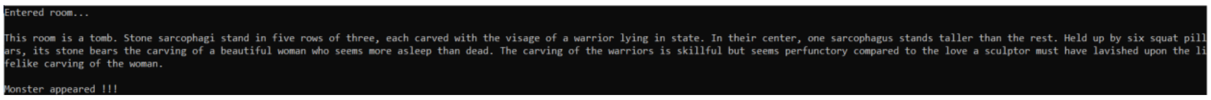
```
        TreeNode<DoublyLinkedList<Room>>* root = new TreeNode<DoublyLinkedList<Room>>();
        // build the tree
        root = tree.insertLevelOrder(arr, root, 0, n, NULL);
        return root;
    }
```

Figure: break point (CodeReuse.h)

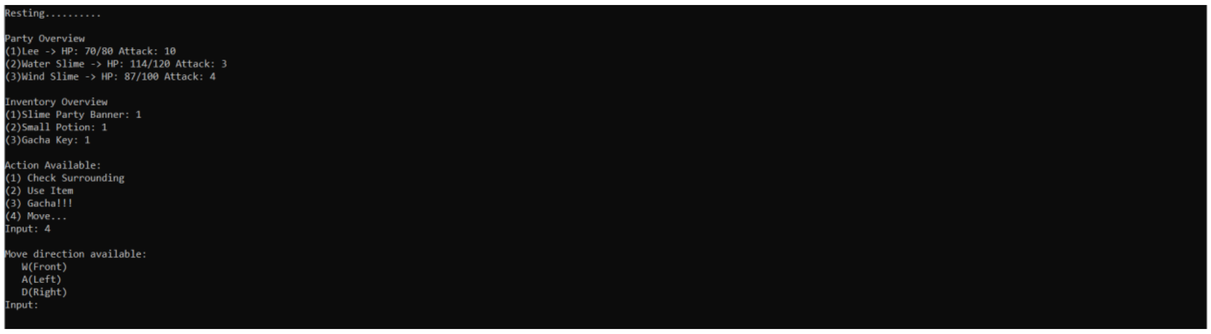


```
Autos
Search (Ctrl+E)    Search Depth: 3
Name                Value                                                                                                                                    Type
◢ arr               0x009bf10c {{size=2 head=0x0107aad8 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0107b7d8 {...} ...} } ...} ...} ...}, ...}   DoublyLinkedList<Room>[9]
  ◢ [0]             {size=2 head=0x0107aad8 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0107b7d8 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
      size          2                                                                                                                                        int
    ◢ head          0x0107aad8 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0107b7d8 {val={fireElem=0 waterElem=...} ...} ...} } ...} ...}       DoublyLinkedNode<Room> *
      ▷ data        {id=0 roomIntro="" monsters={fElements={size=0 head=0x0107b7d8 {val={fireElem=0 waterElem=0 windElem=...} ...} ...} } ...}                   Room
      ◢ next        0x01086360 {data={id=1 roomIntro="A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet a...  DoublyLinkedNode<Room> *
        ◢ data      {id=1 roomIntro="A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet and is so murky that ...  Room
            id      1                                                                                                                                        int
          ▷ roomIntro "A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet and is so murky that you can't se...  std::string
          ▷ monsters {fElements={size=2 head=0x01086638 {val={fireElem=0 waterElem=0 windElem=0 ...} next=0x01086500 {val=...} } ...} }                   Stack<Monster>
            chest   1 '\x1'                                                                                                                                  unsigned char
          ▷ chests  0x0106f0a0 {fElements={size=18 head=0x0106ad58 {val={name="" description="" amount=0 } next=0x01065d50 {...} } ...} }                   Queue<Item> *
        ▷ next      0x010863c8 {data={id=2 roomIntro="Fire crackles and pops in a small cooking fire set in the center of the room. The smoke from a burning r...  DoublyLinkedNode<Room> *
        ▷ prev      0x0107aad8 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0107b7d8 {val={fireElem=0 waterElem=...} ...} ...} } ...} ...}       DoublyLinkedNode<Room> *
      ▷ prev        0x00000000 <NULL>                                                                                                                        DoublyLinkedNode<Room> *
    ▷ tail          0x01081bb0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01081c18 {val={fireElem=0 waterElem=...} ...} ...} } ...} ...}       DoublyLinkedNode<Room> *
  ▷ [1]             {size=2 head=0x01086e58 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x010868a8 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [2]             {size=2 head=0x01088a00 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x010891b8 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [3]             {size=2 head=0x01088e10 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01088e78 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [4]             {size=2 head=0x0108b0e0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0108b3b8 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [5]             {size=2 head=0x0108b830 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0108acd0 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [6]             {size=2 head=0x0108f9d0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0108f3b8 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [7]             {size=2 head=0x0108fa38 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0108f0e0 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  ▷ [8]             {size=2 head=0x0108e378 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x0108e240 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
  n                 9                                                                                                                                        int
◢ root              0x01098298 {data={size=2 head=0x010965b0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01096e38 {...} ...} } ...} ...} ...} ...}  TreeNode<DoublyLinkedList<Room>> *
  ◢ data            {size=2 head=0x010965b0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01096e38 {val={fireElem=...} ...} ...} } ...} ...} ...}  DoublyLinkedList<Room>
      size          2                                                                                                                                        int
    ◢ head          0x010965b0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01096e38 {val={fireElem=0 waterElem=...} ...} ...} } ...} ...}       DoublyLinkedNode<Room> *
      ▷ data        {id=0 roomIntro="" monsters={fElements={size=0 head=0x01096e38 {val={fireElem=0 waterElem=0 windElem=...} ...} ...} } ...}                   Room
      ◢ next        0x01096d68 {data={id=1 roomIntro="A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet a...  DoublyLinkedNode<Room> *
        ◢ data      {id=1 roomIntro="A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet and is so murky that ...  Room
            id      1                                                                                                                                        int
          ▷ roomIntro "A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet and is so murky that you can't se...  std::string
          ▷ monsters {fElements={size=2 head=0x01096a28 {val={fireElem=0 waterElem=0 windElem=0 ...} next=0x01097930 {val=...} } ...} }                   Stack<Monster>
            chest   1 '\x1'                                                                                                                                  unsigned char
          ▷ chests  0x0106f0a0 {fElements={size=18 head=0x0106ad58 {val={name="" description="" amount=0 } next=0x01065d50 {...} } ...} }                   Queue<Item> *
        ▷ next      0x01096888 {data={id=2 roomIntro="Fire crackles and pops in a small cooking fire set in the center of the room. The smoke from a burning r...  DoublyLinkedNode<Room> *
        ▷ prev      0x010965b0 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01096e38 {val={fireElem=0 waterElem=...} ...} ...} } ...} ...}       DoublyLinkedNode<Room> *
      ▷ prev        0x00000000 <NULL>                                                                                                                        DoublyLinkedNode<Room> *
    ▷ tail          0x01096c98 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01096270 {val={fireElem=0 waterElem=...} ...} ...} } ...} ...}       DoublyLinkedNode<Room> *
  ▷ parent          0x00000000 <NULL>                                                                                                                        TreeNode<DoublyLinkedList<Room>> *
  ▷ left            0x010985f8 {data={size=2 head=0x01097860 {data={id=0 roomIntro="" monsters={fElements={size=0 head=0x01097a68 {...} ...} } ...} ...} ...} ...}  TreeNode<DoublyLinkedList<Room>> *
```

Figure: Variable state at break point.



```
Entered room...

This room is a tomb. Stone sarcophagi stand in five rows of three, each carved with the visage of a warrior lying in state. In their center, one sarcophagus stands taller than the rest. Held up by six squat pill
ars, its stone bears the carving of a beautiful woman who seems more asleep than dead. The carving of the warriors is skillful but seems perfunctory compared to the love a sculptor must have lavished upon the li
felike carving of the woman.

Monster appeared !!!
```



```
Resting..........

Party Overview
(1)Lee -> HP: 70/80 Attack: 10
(2)Water Slime -> HP: 114/120 Attack: 3
(3)Wind Slime -> HP: 87/100 Attack: 4

Inventory Overview
(1)Slime Party Banner: 1
(2)Small Potion: 1
(3)Gacha Key: 1

Action Available:
(1) Check Surrounding
(2) Use Item
(3) Gacha!!!
(4) Move...
Input: 4

Move direction available:
   W(Front)
   A(Left)
   D(Right)
Input:
```

Figure: Console output

Troubleshooting

One thing to note is the array will decay into a pointer when passing into a function, and the information of the size of an array need to pass into the function too as there is no additional attribute for the information.

Singly Linked-List

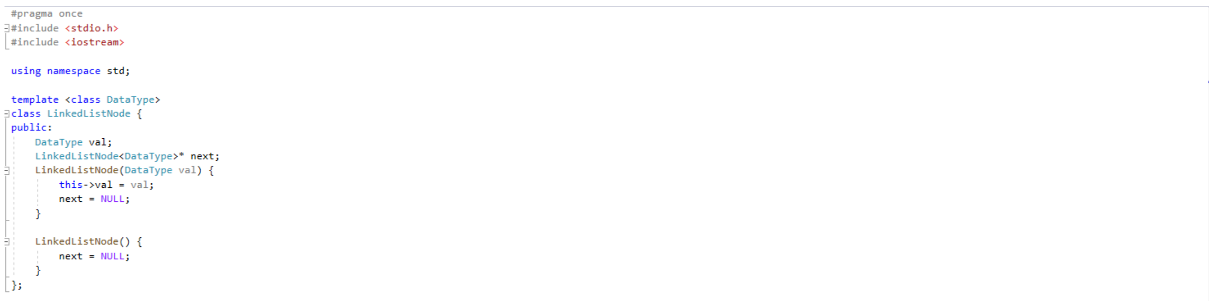
Task Description

In this game, the player can form a party for the adventure. Currently, the number of party members is fixed to be three. However, I would like to allow the adding and removing of party member in the future. Thus, array is not scalable and impractical for this requirement. There is an inventory system for the party too. The inventory may hold up a few items or a lot of items. By used up an item, it will be removed from the inventory and pick up an item will add it to the inventory. I need a data structure that can adapt to the storage requirement dynamically.

Concept

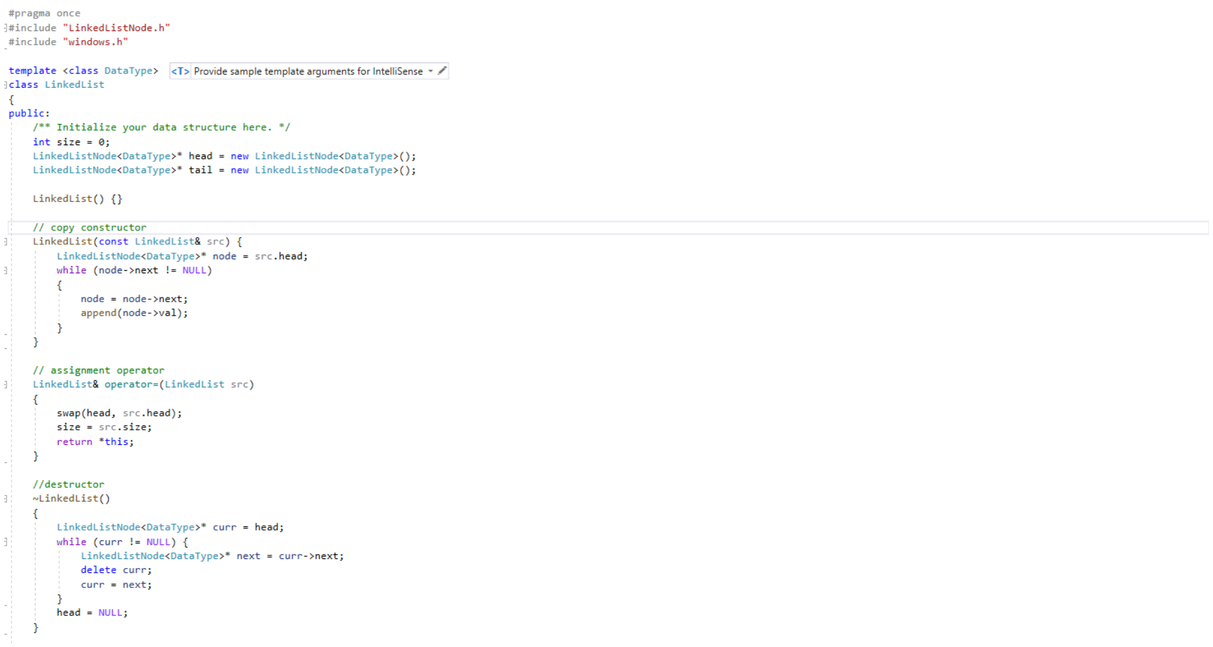
Singly linked list is used to solve this problem. In my implementation, there is head and tail pointer, thus the time complexity to prepend and append is both O(1). The linked list consists solely of data and pointer to the next node. Therefore, only collected item will increase the storage cost of linked list and potential unused space problem in array will not happen. To prepend and append in array, the time complexity is O(n) as it is needed to move all elements in the array to arrange for the position. There is a need to search for certain item in the inventory, and the time complexity for the operation is O(n) for linked list, but it is O(n) for array too, as the position of the item is not known and traverse through the whole array is needed in the worst-case scenario. Frequent delete at certain index operation is carried out when item is used up, and linked-list triumph over array with O(1) time complexity.

Implementation & Output

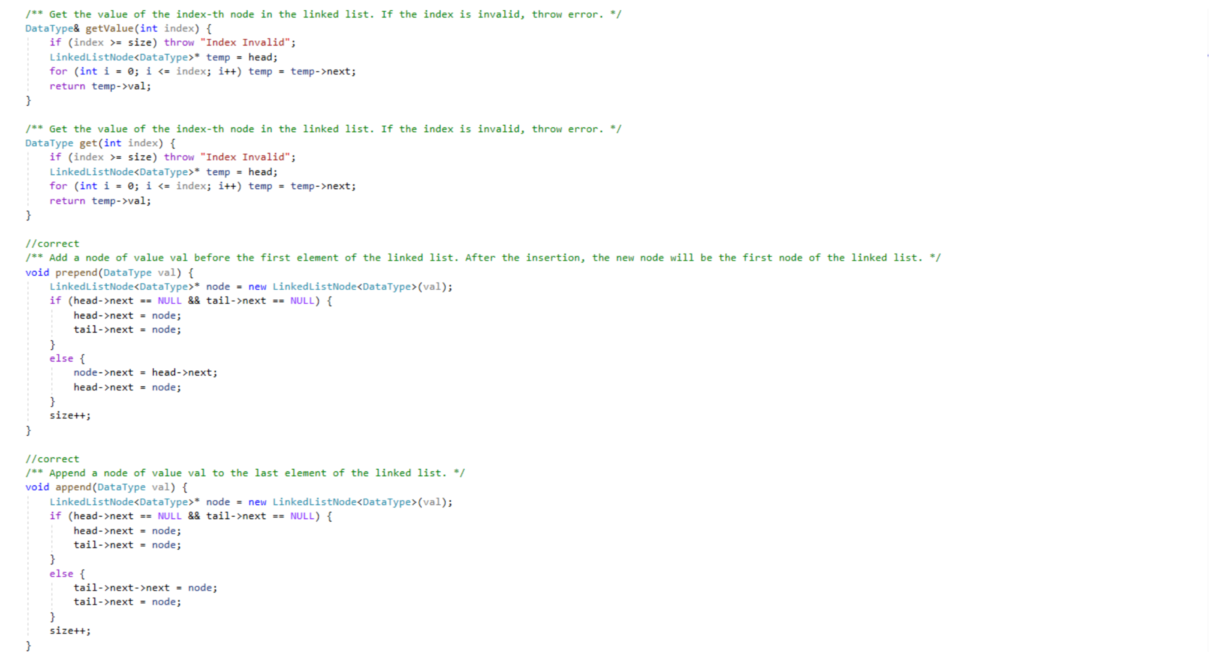


```cpp
#pragma once
#include <stdio.h>
#include <iostream>

using namespace std;

template <class DataType>
class LinkedListNode {
public:
    DataType val;
    LinkedListNode<DataType>* next;
    LinkedListNode(DataType val) {
        this->val = val;
        next = NULL;
    }

    LinkedListNode() {
        next = NULL;
    }
};
```

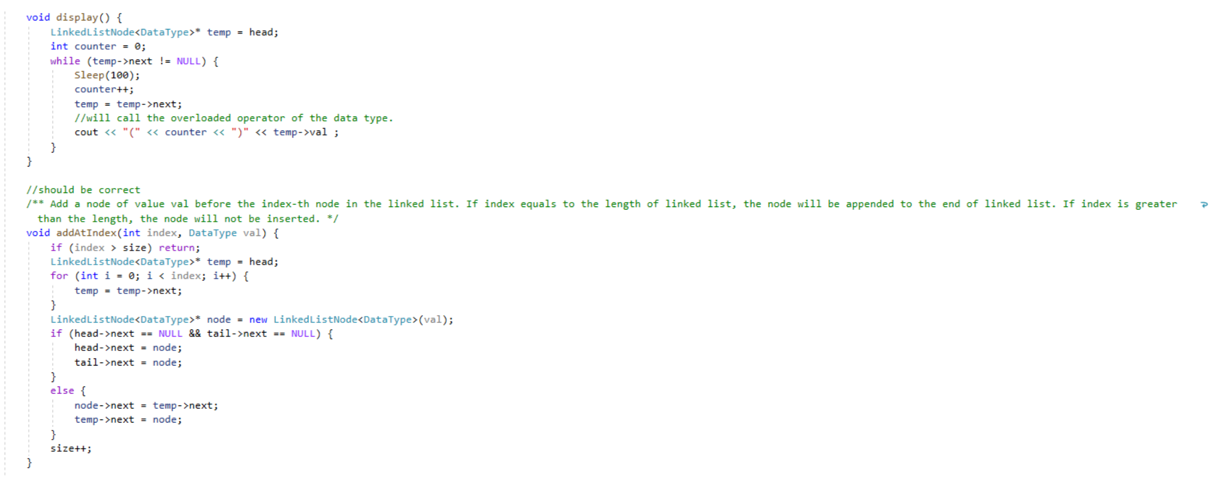
<u>Figure: Node for Linked List, template class for generic purpose, this node will hold a data with type of DataType and a pointer to next node (LinkedListNode.h)</u>



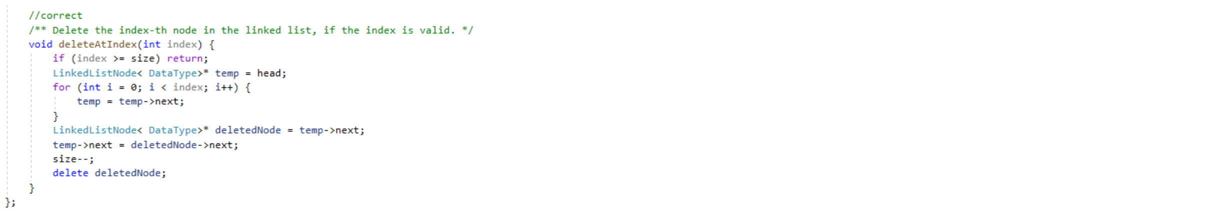
```cpp
#pragma once
#include "LinkedListNode.h"
#include "windows.h"

template <class DataType>  <T> Provide sample template arguments for IntelliSense
class LinkedList
{
public:
    /** Initialize your data structure here. */
    int size = 0;
    LinkedListNode<DataType>* head = new LinkedListNode<DataType>();
    LinkedListNode<DataType>* tail = new LinkedListNode<DataType>();

    LinkedList() {}

    // copy constructor
    LinkedList(const LinkedList& src) {
        LinkedListNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->val);
        }
    }

    // assignment operator
    LinkedList& operator=(LinkedList src)
    {
        swap(head, src.head);
        size = src.size;
        return *this;
    }

    //destructor
    ~LinkedList()
    {
        LinkedListNode<DataType>* curr = head;
        while (curr != NULL) {
            LinkedListNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }
```



```
/** Get the value of the index-th node in the linked list. If the index is invalid, throw error. */
DataType& getValue(int index) {
    if (index >= size) throw "Index Invalid";
    LinkedListNode<DataType>* temp = head;
    for (int i = 0; i <= index; i++) temp = temp->next;
    return temp->val;
}

/** Get the value of the index-th node in the linked list. If the index is invalid, throw error. */
DataType get(int index) {
    if (index >= size) throw "Index Invalid";
    LinkedListNode<DataType>* temp = head;
    for (int i = 0; i <= index; i++) temp = temp->next;
    return temp->val;
}

//correct
/** Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. */
void prepend(DataType val) {
    LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
    if (head->next == NULL && tail->next == NULL) {
        head->next = node;
        tail->next = node;
    }
    else {
        node->next = head->next;
        head->next = node;
    }
    size++;
}

//correct
/** Append a node of value val to the last element of the linked list. */
void append(DataType val) {
    LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
    if (head->next == NULL && tail->next == NULL) {
        head->next = node;
        tail->next = node;
    }
    else {
        tail->next->next = node;
        tail->next = node;
    }
    size++;
}
```

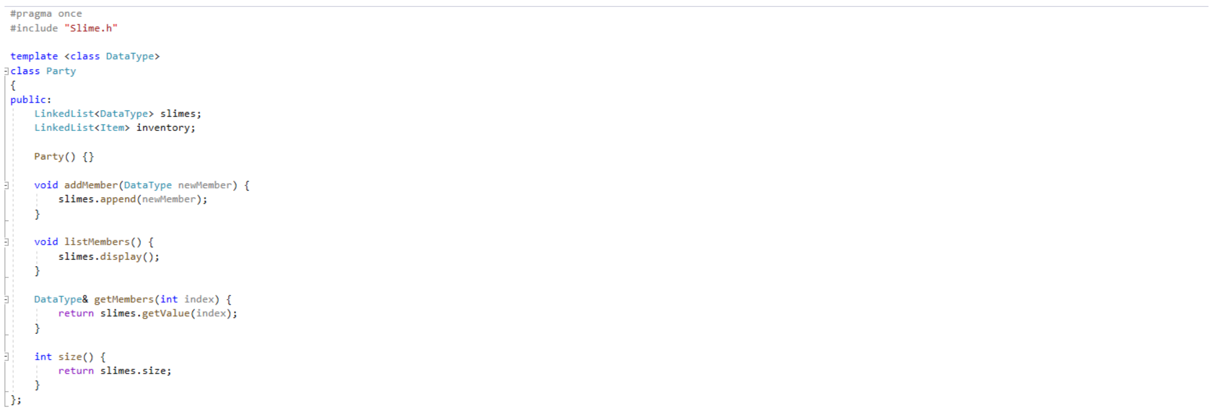


```
void display() {
    LinkedListNode<DataType>* temp = head;
    int counter = 0;
    while (temp->next != NULL) {
        Sleep(100);
        counter++;
        temp = temp->next;
        //will call the overloaded operator of the data type.
        cout << "(" << counter << ")" << temp->val ;
    }
}

//should be correct
/** Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater
  than the length, the node will not be inserted. */
void addAtIndex(int index, DataType val) {
    if (index > size) return;
    LinkedListNode<DataType>* temp = head;
    for (int i = 0; i < index; i++) {
        temp = temp->next;
    }
    LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
    if (head->next == NULL && tail->next == NULL) {
        head->next = node;
        tail->next = node;
    }
    else {
        node->next = temp->next;
        temp->next = node;
    }
    size++;
}
```



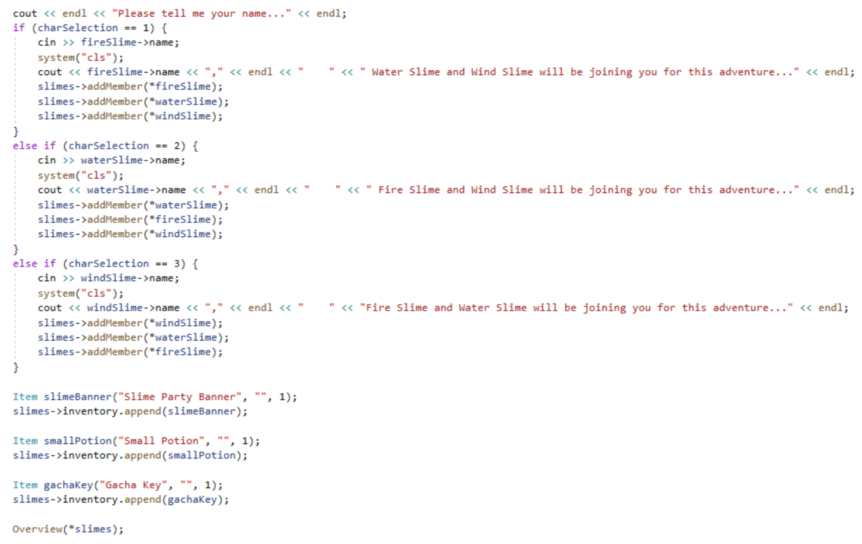
```
    //correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        LinkedListNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode< DataType>* deletedNode = temp->next;
        temp->next = deletedNode->next;
        size--;
        delete deletedNode;
    }
};
```

Figure: Singly linked-list implementation (LinkedList.h)



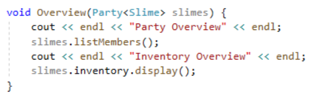
```
#pragma once
#include "Slime.h"

template <class DataType>
class Party
{
public:
    LinkedList<DataType> slimes;
    LinkedList<Item> inventory;

    Party() {}

    void addMember(DataType newMember) {
        slimes.append(newMember);
    }

    void listMembers() {
        slimes.display();
    }

    DataType& getMembers(int index) {
        return slimes.getValue(index);
    }

    int size() {
        return slimes.size;
    }
};
```

Figure: Party class (Party.h)



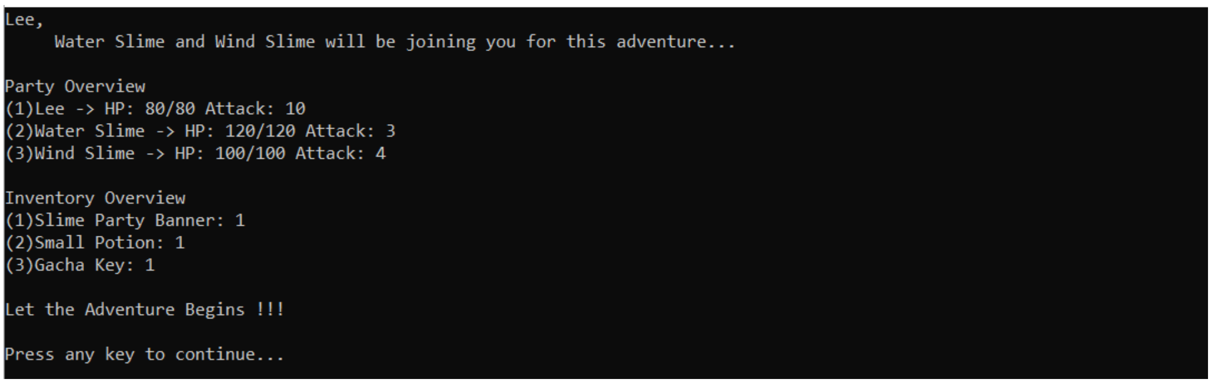
```
cout << endl << "Please tell me your name..." << endl;
if (charSelection == 1) {
    cin >> fireSlime->name;
    system("cls");
    cout << fireSlime->name << "," << endl << "    " << " Water Slime and Wind Slime will be joining you for this adventure..." << endl;
    slimes->addMember(*fireSlime);
    slimes->addMember(*waterSlime);
    slimes->addMember(*windSlime);
}
else if (charSelection == 2) {
    cin >> waterSlime->name;
    system("cls");
    cout << waterSlime->name << "," << endl << "    " << " Fire Slime and Wind Slime will be joining you for this adventure..." << endl;
    slimes->addMember(*waterSlime);
    slimes->addMember(*fireSlime);
    slimes->addMember(*windSlime);
}
else if (charSelection == 3) {
    cin >> windSlime->name;
    system("cls");
    cout << windSlime->name << "," << endl << "    " << "Fire Slime and Water Slime will be joining you for this adventure..." << endl;
    slimes->addMember(*windSlime);
    slimes->addMember(*waterSlime);
    slimes->addMember(*fireSlime);
}

Item slimeBanner("Slime Party Banner", "", 1);
slimes->inventory.append(slimeBanner);

Item smallPotion("Small Potion", "", 1);
slimes->inventory.append(smallPotion);

Item gachaKey("Gacha Key", "", 1);
slimes->inventory.append(gachaKey);

Overview(*slimes);
```

Figure: Append the items into the inventory and add the slimes into party (CodeReuse.h)



```
void Overview(Party<Slime> slimes) {
    cout << endl << "Party Overview" << endl;
    slimes.listMembers();
    cout << endl << "Inventory Overview" << endl;
    slimes.inventory.display();
}
```

Figure: Print the item in inventory and slimes in party (CodeReuse.h)



```
Lee,
     Water Slime and Wind Slime will be joining you for this adventure...

Party Overview
(1)Lee -> HP: 80/80 Attack: 10
(2)Water Slime -> HP: 120/120 Attack: 3
(3)Wind Slime -> HP: 100/100 Attack: 4

Inventory Overview
(1)Slime Party Banner: 1
(2)Small Potion: 1
(3)Gacha Key: 1

Let the Adventure Begins !!!

Press any key to continue...
```

Figure: Console Output

Troubleshooting

The difficulties are mainly due to my perfectionism and paranoid of dynamic storage allocation. I tried to use automatic variable all the time to prevent memory leak. However, dynamic storage allocation is unavoidable for linked list. I could declare linked list as automatic variable, but it still need dynamic storage allocation for data manged by linked list. If copy one linked list to another, both will share the same set of data as the default copy constructor does only shallow copy. As an automatic variable, the destructor will be called when it is out of scope, and this will have multiple delete problem.

Thus, I must implement custom copy constructor to manage the dynamically allocated resource. Thanks to https://stackoverflow.com/questions/3279543/what-is-the-copy-and-swap-idiom , I managed to do it easily. The assignment operator could be implemented easily if the custom copy constructor had been implemented.

Doubly Linked-List

Task Description

The binary tree could handle three directions, namely left, right, and back. The front direction could be achieved through 3-ary tree, but this is will increase the branching factor, and this is not what I want. Moreover, I would like to create a world with different regions e.g. larva room, ice room, rocky room. Thus, I need to change the node contain one room to node contain few rooms and the player can navigate front and back in these rooms and only left, right at the junction to other nodes. In the end, I achieved this by using a doubly linked list to two rooms and each node is made up of the doubly linked list.

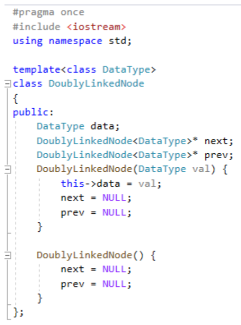
Concept

In singly linked list, only one direction is available when traversing through the list but navigate to the previous room is required. Doubly linked list could fulfil this requirement by just adding one pointer pointing to the previous node. The other modification to the code is the correct assignment of the pointer pointing to the previous node when carry out adding and deletion of node.

In my implementation, there is head and tail pointer, thus the time complexity to prepend and append is both O(1). To prepend and append in array, the time complexity is O(n) as it is needed to move all elements in the array to arrange for the position.

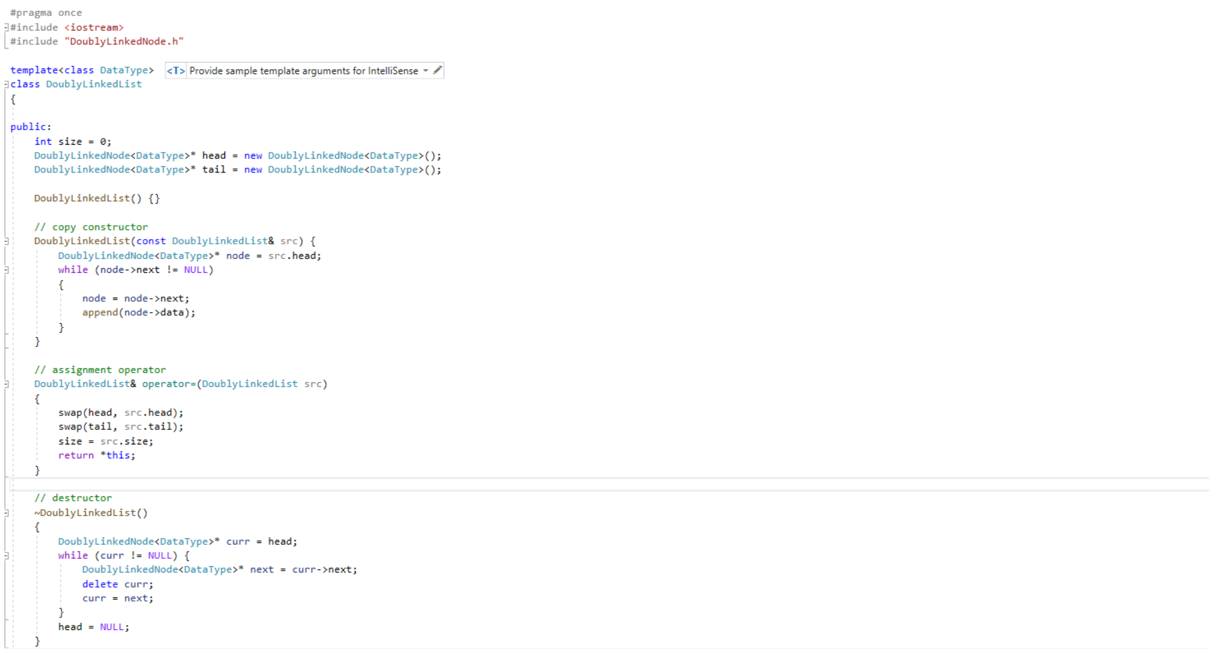
The doubly linked list consists solely of data, a pointer to next node, and a pointer to previous node. Therefore, the client code only needs to store the current node, instead of the whole array to get the previous/next node if array is used. In the future, I could generate random number of rooms without worry of limited space or unused space.

Implementation & Output



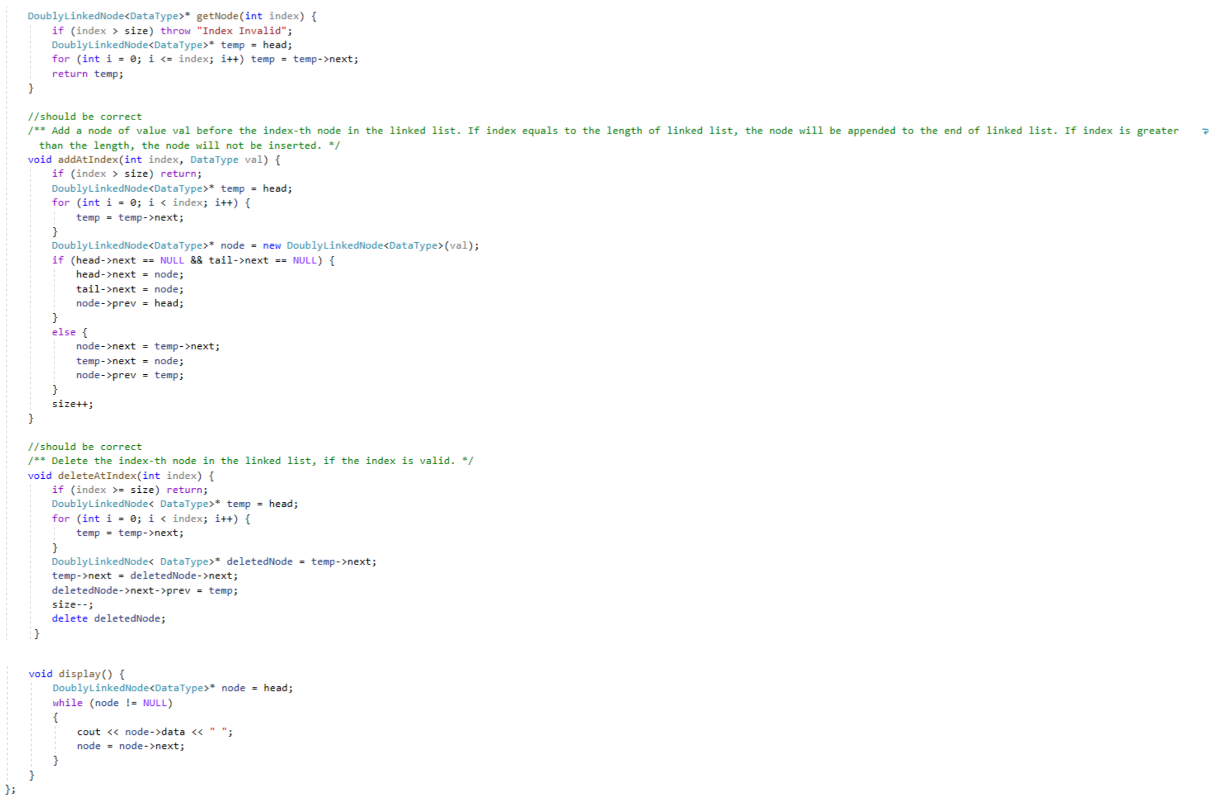
```
#pragma once
#include <iostream>
using namespace std;

template<class DataType>
class DoublyLinkedNode
{
public:
    DataType data;
    DoublyLinkedNode<DataType>* next;
    DoublyLinkedNode<DataType>* prev;
    DoublyLinkedNode(DataType val) {
        this->data = val;
        next = NULL;
        prev = NULL;
    }

    DoublyLinkedNode() {
        next = NULL;
        prev = NULL;
    }
};
```

Figure: Doubly linked node (DoublyLinkedNode.h)



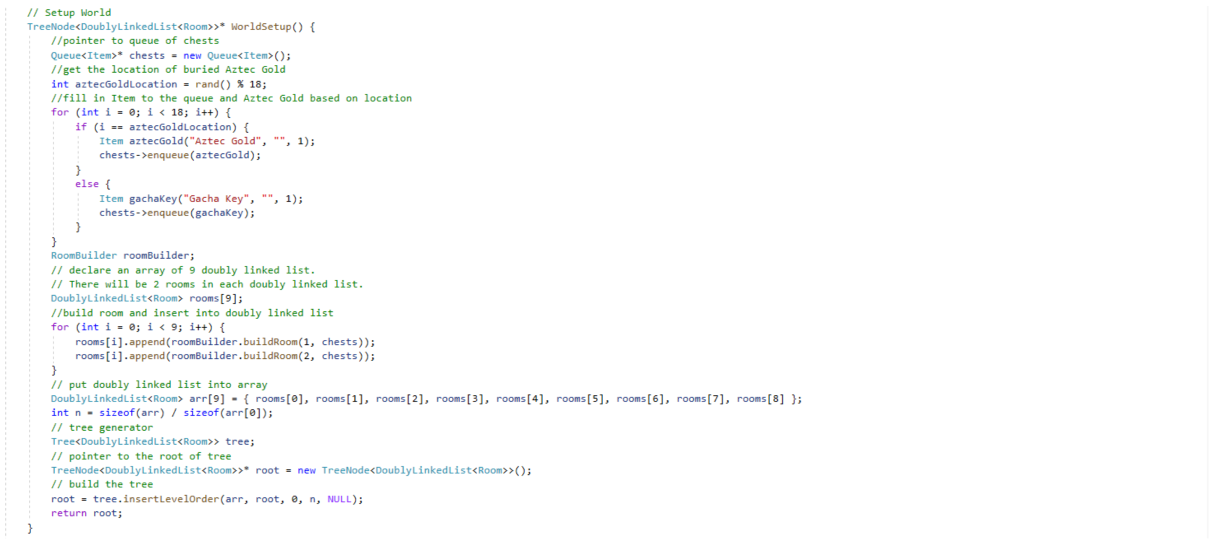
```cpp
#pragma once
#include <iostream>
#include "DoublyLinkedNode.h"

template<class DataType>  <T> Provide sample template arguments for IntelliSense
class DoublyLinkedList
{

public:
    int size = 0;
    DoublyLinkedNode<DataType>* head = new DoublyLinkedNode<DataType>();
    DoublyLinkedNode<DataType>* tail = new DoublyLinkedNode<DataType>();

    DoublyLinkedList() {}

    // copy constructor
    DoublyLinkedList(const DoublyLinkedList& src) {
        DoublyLinkedNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->data);
        }
    }

    // assignment operator
    DoublyLinkedList& operator=(DoublyLinkedList src)
    {
        swap(head, src.head);
        swap(tail, src.tail);
        size = src.size;
        return *this;
    }

    // destructor
    ~DoublyLinkedList()
    {
        DoublyLinkedNode<DataType>* curr = head;
        while (curr != NULL) {
            DoublyLinkedNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }
```

```cpp
    // add at the start
    void prepend(DataType val)
    {
        DoublyLinkedNode<DataType>* node = new DoublyLinkedNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
            node->prev = head;
        }
        else {
            node->prev = head;
            node->next = head->next;
            head->next = node;
        }
        size++;
    }

   // add at the end
    void append(DataType val)
    {
        DoublyLinkedNode<DataType>* node = new DoublyLinkedNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
            node->prev = head;
        }
        else {
            node->prev = tail->next;
            tail->next->next = node;
            tail->next = node;
        }
        size++;
    }

    /* Get the value of the index-th node in the linked list. If the index is invalid, throw error. */
    DataType& get(int index) {
        if (index > size) throw "Index Invalid";
        DoublyLinkedNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }
```



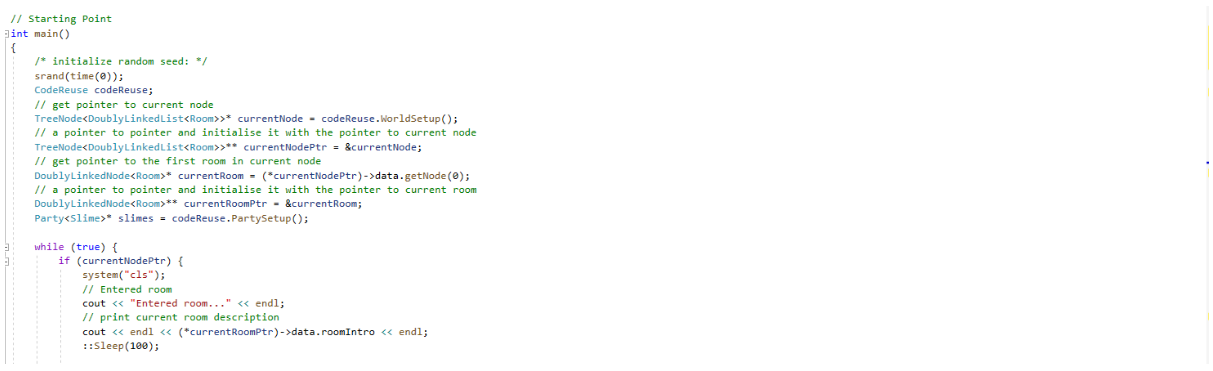
```
    DoublyLinkedNode<DataType>* getNode(int index) {
        if (index > size) throw "Index Invalid";
        DoublyLinkedNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp;
    }

    //should be correct
    /** Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater
      than the length, the node will not be inserted. */
    void addAtIndex(int index, DataType val) {
        if (index > size) return;
        DoublyLinkedNode<DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        DoublyLinkedNode<DataType>* node = new DoublyLinkedNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
            node->prev = head;
        }
        else {
            node->next = temp->next;
            temp->next = node;
            node->prev = temp;
        }
        size++;
    }

    //should be correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        DoublyLinkedNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        DoublyLinkedNode< DataType>* deletedNode = temp->next;
        temp->next = deletedNode->next;
        deletedNode->next->prev = temp;
        size--;
        delete deletedNode;
     }


    void display() {
        DoublyLinkedNode<DataType>* node = head;
        while (node != NULL)
        {
            cout << node->data << " ";
            node = node->next;
        }
    }
};
```

Figure: Doubly Linked List (DoublyLinkedList.h)



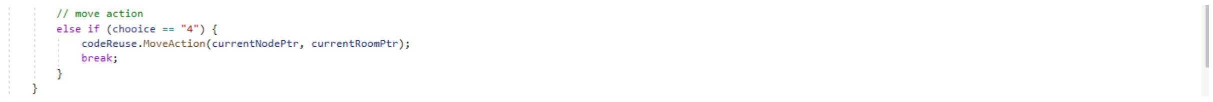
```
    // Setup World
    TreeNode<DoublyLinkedList<Room>>* WorldSetup() {
        //pointer to queue of chests
        Queue<Item>* chests = new Queue<Item>();
        //get the location of buried Aztec Gold
        int aztecGoldLocation = rand() % 18;
        //fill in Item to the queue and Aztec Gold based on location
        for (int i = 0; i < 18; i++) {
            if (i == aztecGoldLocation) {
                Item aztecGold("Aztec Gold", "", 1);
                chests->enqueue(aztecGold);
            }
            else {
                Item gachaKey("Gacha Key", "", 1);
                chests->enqueue(gachaKey);
            }
        }
        RoomBuilder roomBuilder;
        // declare an array of 9 doubly linked list.
        // There will be 2 rooms in each doubly linked list.
        DoublyLinkedList<Room> rooms[9];
        //build room and insert into doubly linked list
        for (int i = 0; i < 9; i++) {
            rooms[i].append(roomBuilder.buildRoom(1, chests));
            rooms[i].append(roomBuilder.buildRoom(2, chests));
        }
        // put doubly linked list into array
        DoublyLinkedList<Room> arr[9] = { rooms[0], rooms[1], rooms[2], rooms[3], rooms[4], rooms[5], rooms[6], rooms[7], rooms[8] };
        int n = sizeof(arr) / sizeof(arr[0]);
        // tree generator
        Tree<DoublyLinkedList<Room>> tree;
        // pointer to the root of tree
        TreeNode<DoublyLinkedList<Room>>* root = new TreeNode<DoublyLinkedList<Room>>();
        // build the tree
        root = tree.insertLevelOrder(arr, root, 0, n, NULL);
        return root;
    }
```

Figure: pass the doubly linked list into array and generate the tree with one node consists of one doubly linked list (CodeReuse.h)



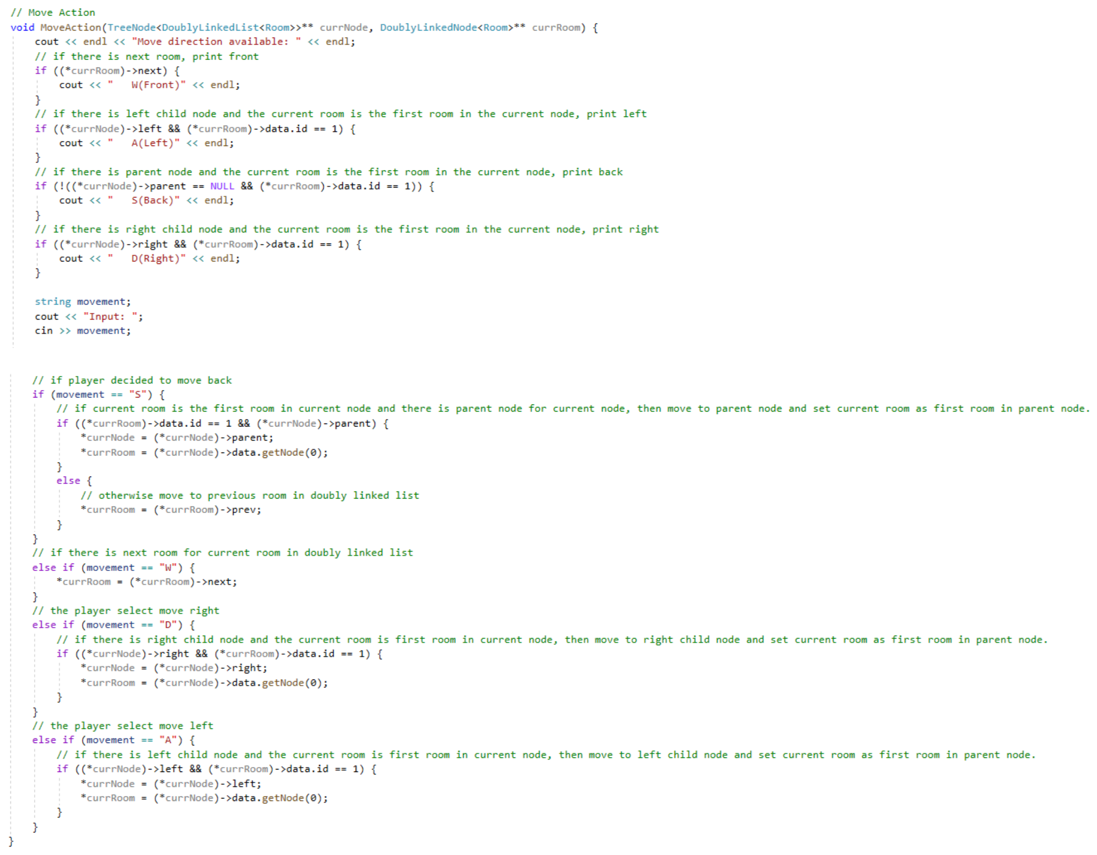
```cpp
// Starting Point
int main()
{
    /* initialize random seed: */
    srand(time(0));
    CodeReuse codeReuse;
    // get pointer to current node
    TreeNode<DoublyLinkedList<Room>>* currentNode = codeReuse.WorldSetup();
    // a pointer to pointer and initialise it with the pointer to current node
    TreeNode<DoublyLinkedList<Room>>** currentNodePtr = &currentNode;
    // get pointer to the first room in current node
    DoublyLinkedNode<Room>* currentRoom = (*currentNodePtr)->data.getNode(0);
    // a pointer to pointer and initialise it with the pointer to current room
    DoublyLinkedNode<Room>** currentRoomPtr = &currentRoom;
    Party<Slime>* slimes = codeReuse.PartySetup();

    while (true) {
        if (currentNodePtr) {
            system("cls");
            // Entered room
            cout << "Entered room..." << endl;
            // print current room description
            cout << endl << (*currentRoomPtr)->data.roomIntro << endl;
            ::Sleep(100);
```

Figure: get root node and set the first room as current room, later print the description of current room (main.cpp)



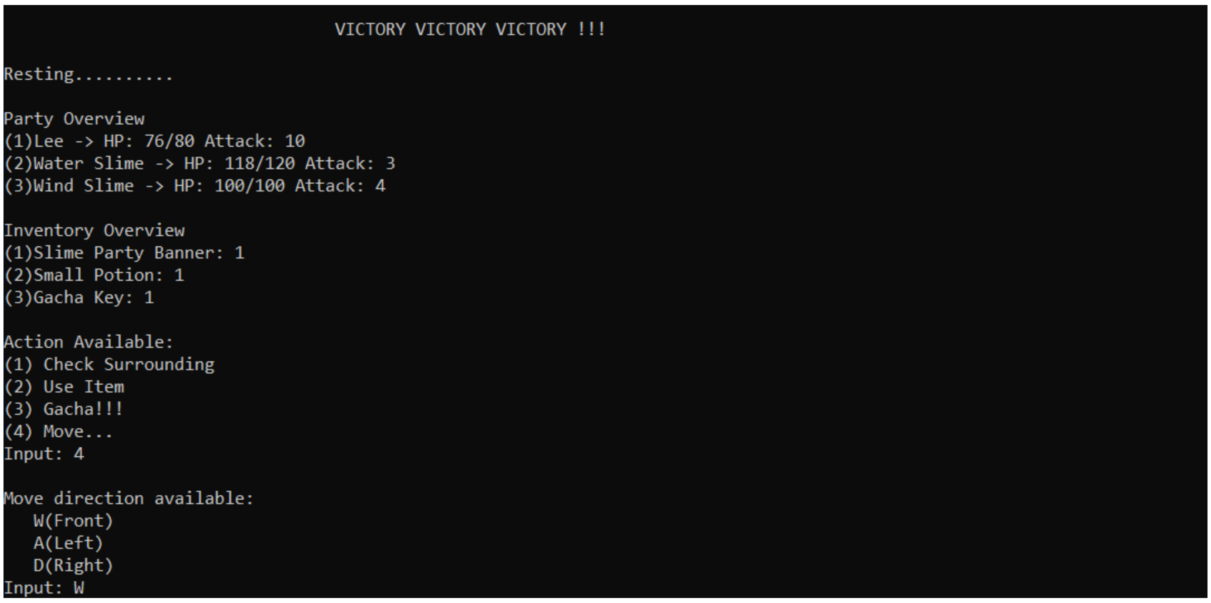
```cpp
            // move action
            else if (chooice == "4") {
                codeReuse.MoveAction(currentNodePtr, currentRoomPtr);
                break;
            }
    }
```

Figure: call MoveAction function if player decide to go to other room (main.cpp)

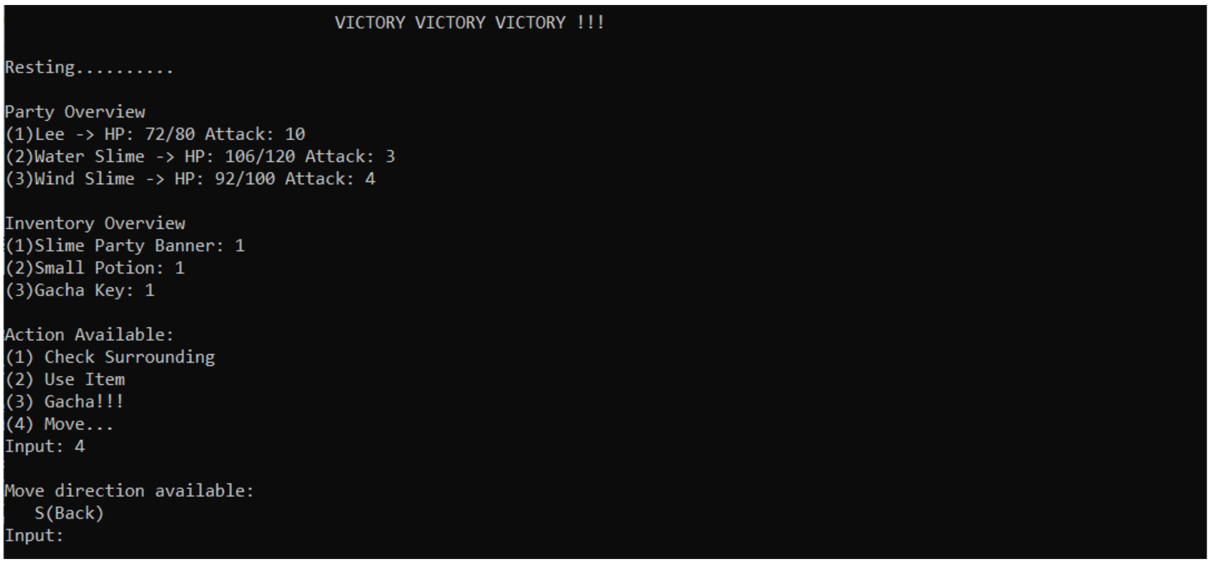


```cpp
// Move Action
void MoveAction(TreeNode<DoublyLinkedList<Room>>** currNode, DoublyLinkedNode<Room>** currRoom) {
    cout << endl << "Move direction available: " << endl;
    // if there is next room, print front
    if ((*currRoom)->next) {
        cout << "   W(Front)" << endl;
    }
    // if there is left child node and the current room is the first room in the current node, print left
    if ((*currNode)->left && (*currRoom)->data.id == 1) {
        cout << "   A(Left)" << endl;
    }
    // if there is parent node and the current room is the first room in the current node, print back
    if (!((*currNode)->parent == NULL && (*currRoom)->data.id == 1)) {
        cout << "   S(Back)" << endl;
    }
    // if there is right child node and the current room is the first room in the current node, print right
    if ((*currNode)->right && (*currRoom)->data.id == 1) {
        cout << "   D(Right)" << endl;
    }

    string movement;
    cout << "Input: ";
    cin >> movement;


    // if player decided to move back
    if (movement == "S") {
        // if current room is the first room in current node and there is parent node for current node, then move to parent node and set current room as first room in parent node.
        if ((*currRoom)->data.id == 1 && (*currNode)->parent) {
            *currNode = (*currNode)->parent;
            *currRoom = (*currNode)->data.getNode(0);
        }
        else {
            // otherwise move to previous room in doubly linked list
            *currRoom = (*currRoom)->prev;
        }
    }
    // if there is next room for current room in doubly linked list
    else if (movement == "W") {
        *currRoom = (*currRoom)->next;
    }
    // the player select move right
    else if (movement == "D") {
        // if there is right child node and the current room is first room in current node, then move to right child node and set current room as first room in parent node.
        if ((*currNode)->right && (*currRoom)->data.id == 1) {
            *currNode = (*currNode)->right;
            *currRoom = (*currNode)->data.getNode(0);
        }
    }
    // the player select move left
    else if (movement == "A") {
        // if there is left child node and the current room is first room in current node, then move to left child node and set current room as first room in parent node.
        if ((*currNode)->left && (*currRoom)->data.id == 1) {
            *currNode = (*currNode)->left;
            *currRoom = (*currNode)->data.getNode(0);
        }
    }
}
```

Figure: MoveAction function (CodeReuse.h)



```
                                 VICTORY VICTORY VICTORY !!!

Resting..........

Party Overview
(1)Lee -> HP: 76/80 Attack: 10
(2)Water Slime -> HP: 118/120 Attack: 3
(3)Wind Slime -> HP: 100/100 Attack: 4

Inventory Overview
(1)Slime Party Banner: 1
(2)Small Potion: 1
(3)Gacha Key: 1

Action Available:
(1) Check Surrounding
(2) Use Item
(3) Gacha!!!
(4) Move...
Input: 4

Move direction available:
   W(Front)
   A(Left)
   D(Right)
Input: W
```



```
                                 VICTORY VICTORY VICTORY !!!

Resting..........

Party Overview
(1)Lee -> HP: 72/80 Attack: 10
(2)Water Slime -> HP: 106/120 Attack: 3
(3)Wind Slime -> HP: 92/100 Attack: 4

Inventory Overview
(1)Slime Party Banner: 1
(2)Small Potion: 1
(3)Gacha Key: 1

Action Available:
(1) Check Surrounding
(2) Use Item
(3) Gacha!!!
(4) Move...
Input: 4

Move direction available:
   S(Back)
Input:
```

Figure: moving into next room of current room in same node, lead to only back direction available when reached (Console Output)

Troubleshooting

Some nasty problem had been encountered when I was trying to implement the destructor for doubly linked list. It is a multiple delete problem and only exists when I declare the doubly linked list as an automatic variable, instead of dynamically allocated on heap. When it is dynamically allocated, there will be no automatic garbage collection and will lead to memory leak if the programmer does not carry it out manually. However, an automatic variable will be deleted automatically and the destructor will be called. The program halt and show that it is the destructor that caused the problem.

After multiple checks, the code to delete elements in doubly linked list is correct, but the problem lies in I did not implement the copy constructor. With default copy constructor, the value of the pointer is copied. Thus, the first destructor call will be successful, but the second will not as it is trying to delete something non-existed. The problem solved after I performed a deep copy of the elements when the copy constructor is invoked.

Stack

Task Description

While carrying out the adventure in search of Aztec Gold, there will be monsters obstructing the journey. In each room, a few numbers of monsters will be generated. Some monster will have the ability to reincarnate, thus a simple array could not accomplish this task.

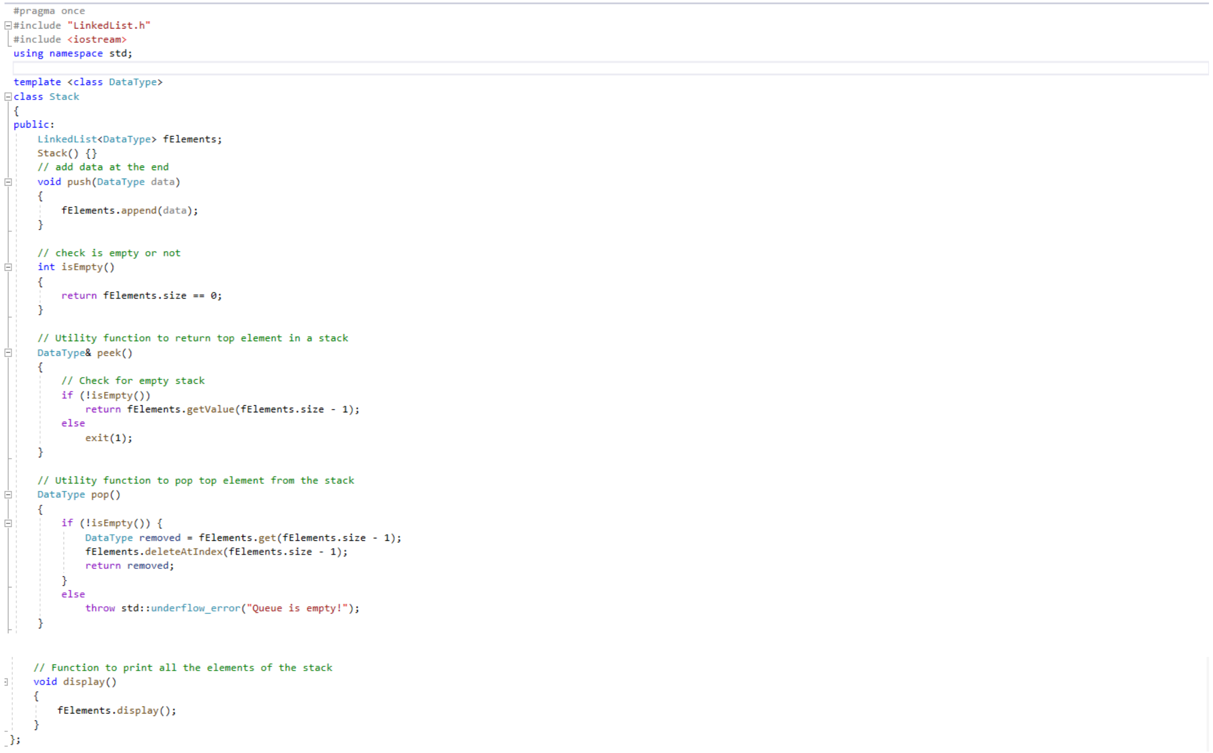
Concept

First, this could be accomplished with a linked list. However, good programs use abstraction and using abstract data types is a better practice. The idea of an Abstract Data Types is the separation of specification (what stuff we are working with and what operations are available) and implementation (the actual implementation of the stuff and its operations). This brings the benefit of understandable code, implementation of ADT can be changed without changes to client code and reusing of ADT in the future.

Stack is first in last out and has three main operations, namely push (insert data), pop (remove data at the top), and peek (get the top element). In my implementation, I use singly linked list as the underlying storage, it's append operations for push, delete at last index operations for pop, and get the element at last index for peek. All of these operations have O(1) time complexity.

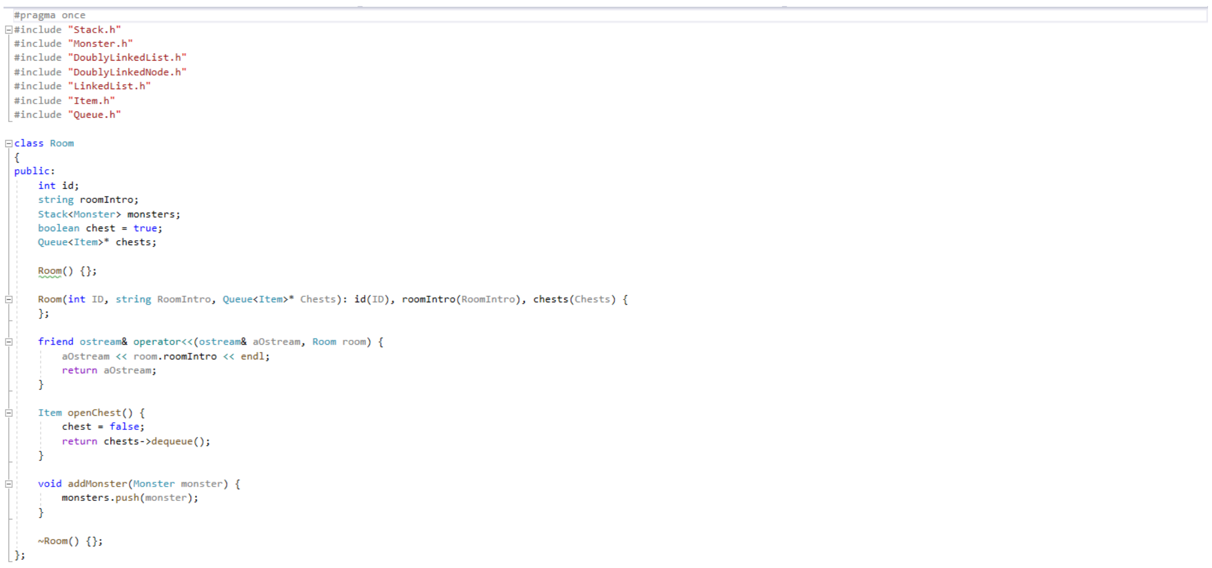
When building the room, the generated monster will be pushed into the stack. In the battle phase, if the stack is not empty, means there are monster in the room. First, peek to get the monster. Pop the monster if it gets defeated. However, there will be a new monster push into the stack if the monster reincarnated and the next battle will be with this monster. The player must empty the stack to proceed.

Implementation & Output



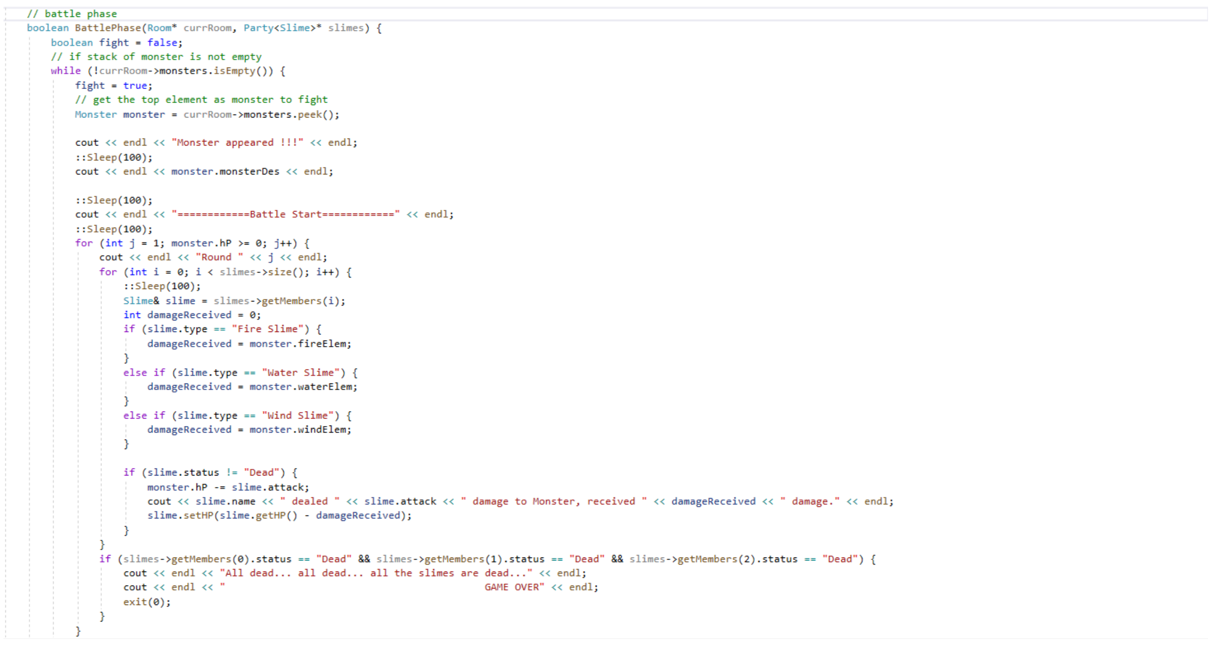
```cpp
#pragma once
#include "LinkedList.h"
#include <iostream>
using namespace std;

template <class DataType>
class Stack
{
public:
    LinkedList<DataType> fElements;
    Stack() {}
    // add data at the end
    void push(DataType data)
    {
        fElements.append(data);
    }

    // check is empty or not
    int isEmpty()
    {
        return fElements.size == 0;
    }

    // Utility function to return top element in a stack
    DataType& peek()
    {
        // Check for empty stack
        if (!isEmpty())
            return fElements.getValue(fElements.size - 1);
        else
            exit(1);
    }

    // Utility function to pop top element from the stack
    DataType pop()
    {
        if (!isEmpty()) {
            DataType removed = fElements.get(fElements.size - 1);
            fElements.deleteAtIndex(fElements.size - 1);
            return removed;
        }
        else
            throw std::underflow_error("Queue is empty!");
    }

    // Function to print all the elements of the stack
    void display()
    {
        fElements.display();
    }
};
```

<u>Figure: Stack implementation with linked list (Stack.h)</u>

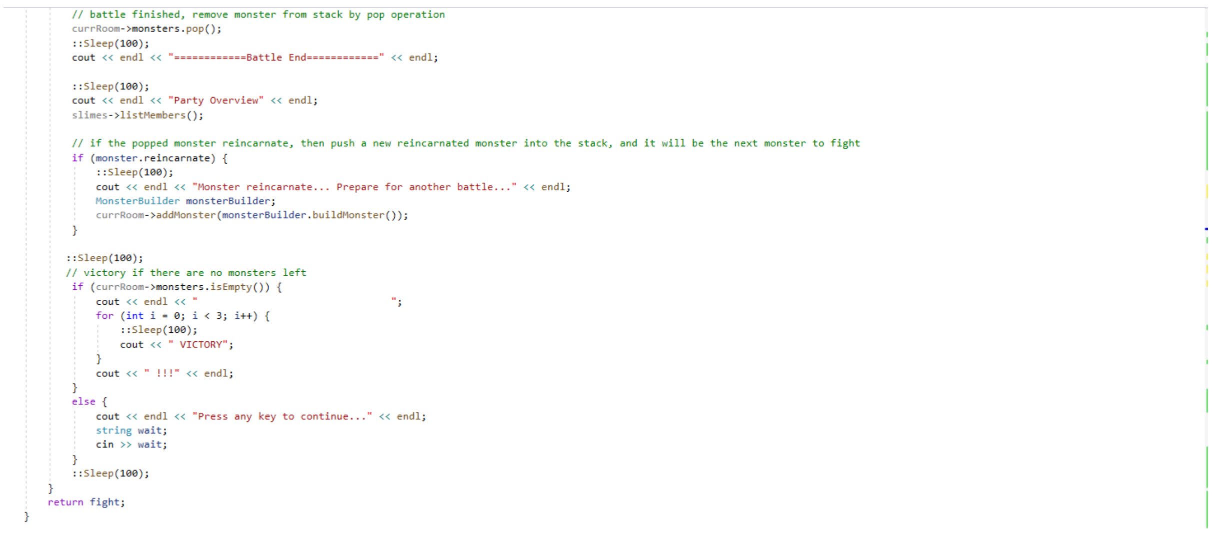


```cpp
#pragma once
#include "Stack.h"
#include "Monster.h"
#include "DoublyLinkedList.h"
#include "DoublyLinkedNode.h"
#include "LinkedList.h"
#include "Item.h"
#include "Queue.h"

class Room
{
public:
    int id;
    string roomIntro;
    Stack<Monster> monsters;
    boolean chest = true;
    Queue<Item>* chests;

    Room() {};

    Room(int ID, string RoomIntro, Queue<Item>* Chests): id(ID), roomIntro(RoomIntro), chests(Chests) {
    };

    friend ostream& operator<<(ostream& aOstream, Room room) {
        aOstream << room.roomIntro << endl;
        return aOstream;
    }

    Item openChest() {
        chest = false;
        return chests->dequeue();
    }

    void addMonster(Monster monster) {
        monsters.push(monster);
    }

    ~Room() {};
};
```

Figure: add monster to the room by push it into stack (Room.h)

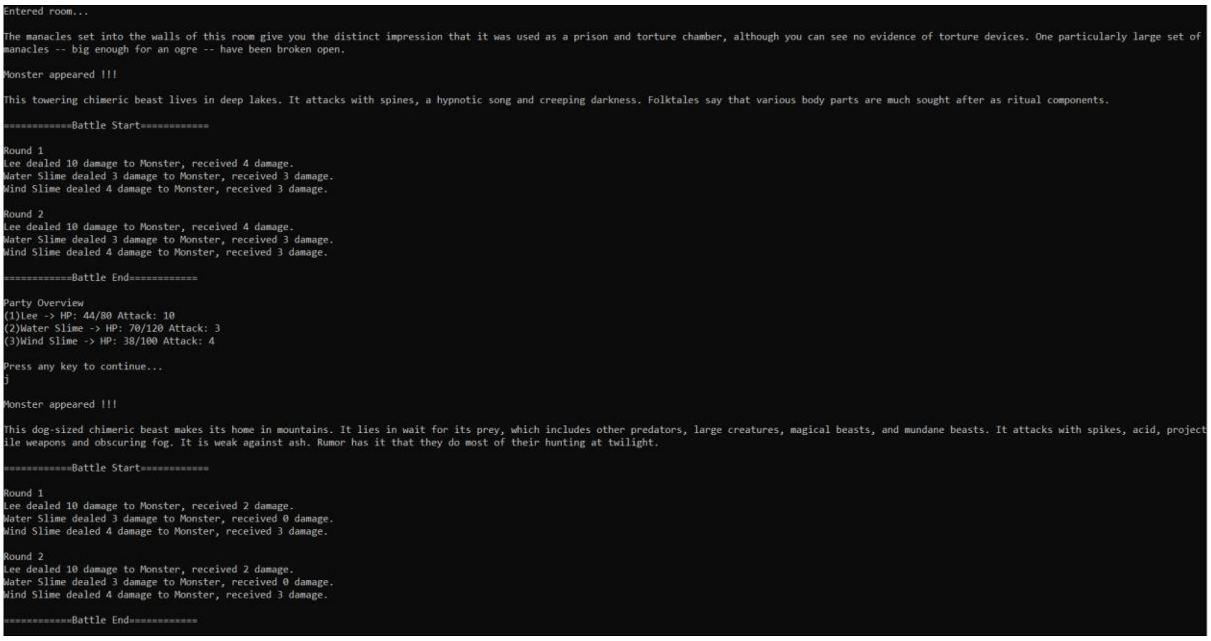


```cpp
    // battle phase
    boolean BattlePhase(Room* currRoom, Party<Slime>* slimes) {
        boolean fight = false;
        // if stack of monster is not empty
        while (!currRoom->monsters.isEmpty()) {
            fight = true;
            // get the top element as monster to fight
            Monster monster = currRoom->monsters.peek();

            cout << endl << "Monster appeared !!!" << endl;
            ::Sleep(100);
            cout << endl << monster.monsterDes << endl;

            ::Sleep(100);
            cout << endl << "============Battle Start============" << endl;
            ::Sleep(100);
            for (int j = 1; monster.hP >= 0; j++) {
                cout << endl << "Round " << j << endl;
                for (int i = 0; i < slimes->size(); i++) {
                    ::Sleep(100);
                    Slime& slime = slimes->getMembers(i);
                    int damageReceived = 0;
                    if (slime.type == "Fire Slime") {
                        damageReceived = monster.fireElem;
                    }
                    else if (slime.type == "Water Slime") {
                        damageReceived = monster.waterElem;
                    }
                    else if (slime.type == "Wind Slime") {
                        damageReceived = monster.windElem;
                    }

                    if (slime.status != "Dead") {
                        monster.hP -= slime.attack;
                        cout << slime.name << " dealed " << slime.attack << " damage to Monster, received " << damageReceived << " damage." << endl;
                        slime.setHP(slime.getHP() - damageReceived);
                    }
                }
                if (slimes->getMembers(0).status == "Dead" && slimes->getMembers(1).status == "Dead" && slimes->getMembers(2).status == "Dead") {
                    cout << endl << "All dead... all dead... all the slimes are dead..." << endl;
                    cout << endl << "                                             GAME OVER" << endl;
                    exit(0);
                }
            }
```

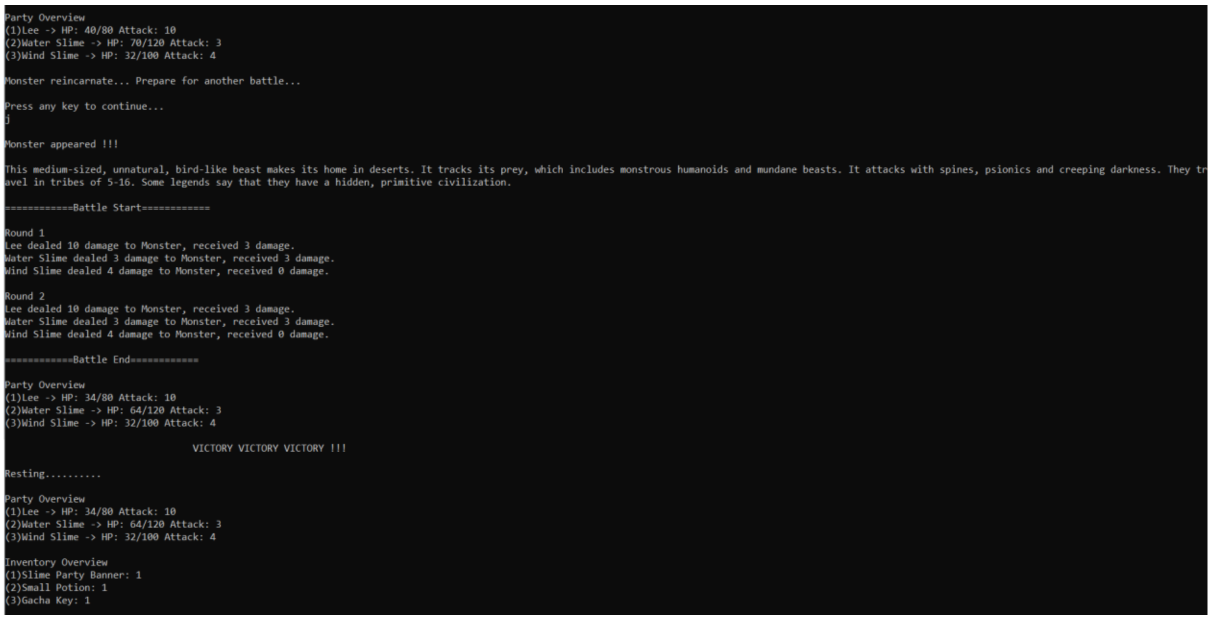


```cpp
            // battle finished, remove monster from stack by pop operation
            currRoom->monsters.pop();
            ::Sleep(100);
            cout << endl << "============Battle End============" << endl;

            ::Sleep(100);
            cout << endl << "Party Overview" << endl;
            slimes->listMembers();

            // if the popped monster reincarnate, then push a new reincarnated monster into the stack, and it will be the next monster to fight
            if (monster.reincarnate) {
                ::Sleep(100);
                cout << endl << "Monster reincarnate... Prepare for another battle..." << endl;
                MonsterBuilder monsterBuilder;
                currRoom->addMonster(monsterBuilder.buildMonster());
            }

            ::Sleep(100);
            // victory if there are no monsters left
            if (currRoom->monsters.isEmpty()) {
                cout << endl << "                                    ";
                for (int i = 0; i < 3; i++) {
                    ::Sleep(100);
                    cout << " VICTORY";
                }
                cout << " !!!" << endl;
            }
            else {
                cout << endl << "Press any key to continue..." << endl;
                string wait;
                cin >> wait;
            }
            ::Sleep(100);
        }
        return fight;
    }
```

Figure: battle phase (CodeReuse.h)



```
Entered room...

The manacles set into the walls of this room give you the distinct impression that it was used as a prison and torture chamber, although you can see no evidence of torture devices. One particularly large set of
manacles -- big enough for an ogre -- have been broken open.

Monster appeared !!!

This towering chimeric beast lives in deep lakes. It attacks with spines, a hypnotic song and creeping darkness. Folktales say that various body parts are much sought after as ritual components.

============Battle Start============

Round 1
Lee dealed 10 damage to Monster, received 4 damage.
Water Slime dealed 3 damage to Monster, received 3 damage.
Wind Slime dealed 4 damage to Monster, received 3 damage.

Round 2
Lee dealed 10 damage to Monster, received 4 damage.
Water Slime dealed 3 damage to Monster, received 3 damage.
Wind Slime dealed 4 damage to Monster, received 3 damage.

============Battle End============

Party Overview
(1)Lee -> HP: 44/80 Attack: 10
(2)Water Slime -> HP: 70/120 Attack: 3
(3)Wind Slime -> HP: 38/100 Attack: 4

Press any key to continue...
j

Monster appeared !!!

This dog-sized chimeric beast makes its home in mountains. It lies in wait for its prey, which includes other predators, large creatures, magical beasts, and mundane beasts. It attacks with spikes, acid, project
ile weapons and obscuring fog. It is weak against ash. Rumor has it that they do most of their hunting at twilight.

============Battle Start============

Round 1
Lee dealed 10 damage to Monster, received 2 damage.
Water Slime dealed 3 damage to Monster, received 0 damage.
Wind Slime dealed 4 damage to Monster, received 3 damage.

Round 2
Lee dealed 10 damage to Monster, received 2 damage.
Water Slime dealed 3 damage to Monster, received 0 damage.
Wind Slime dealed 4 damage to Monster, received 3 damage.

============Battle End============
```



```
Party Overview
(1)Lee -> HP: 40/80 Attack: 10
(2)Water Slime -> HP: 70/120 Attack: 3
(3)Wind Slime -> HP: 32/100 Attack: 4

Monster reincarnate... Prepare for another battle...

Press any key to continue...
j

Monster appeared !!!

This medium-sized, unnatural, bird-like beast makes its home in deserts. It tracks its prey, which includes monstrous humanoids and mundane beasts. It attacks with spines, psionics and creeping darkness. They tr
avel in tribes of 5-16. Some legends say that they have a hidden, primitive civilization.

============Battle Start============

Round 1
Lee dealed 10 damage to Monster, received 3 damage.
Water Slime dealed 3 damage to Monster, received 3 damage.
Wind Slime dealed 4 damage to Monster, received 0 damage.

Round 2
Lee dealed 10 damage to Monster, received 3 damage.
Water Slime dealed 3 damage to Monster, received 3 damage.
Wind Slime dealed 4 damage to Monster, received 0 damage.

============Battle End============

Party Overview
(1)Lee -> HP: 34/80 Attack: 10
(2)Water Slime -> HP: 64/120 Attack: 3
(3)Wind Slime -> HP: 32/100 Attack: 4

                                VICTORY VICTORY VICTORY !!!

Resting..........

Party Overview
(1)Lee -> HP: 34/80 Attack: 10
(2)Water Slime -> HP: 64/120 Attack: 3
(3)Wind Slime -> HP: 32/100 Attack: 4

Inventory Overview
(1)Slime Party Banner: 1
(2)Small Potion: 1
(3)Gacha Key: 1
```

Figure: the room has 2 monsters initially, but one reincarnate, so total 3 battles (Console Output)

Troubleshooting

A problem arises when I changed the stack of monsters from dynamic storage to automatic variable (whether its creation is on heap or stack is determined by the context in which the object is defined, in this case, it is on the stack as the room object is automatic variable too). This is due to the buggy overloaded assignment operator. As it is an automatic variable, I need to use assignment operator when returning the data of monster from called function to calling function. In the implementation of the assignment operator, I forgot to initialise the size attribute which indicates the number of monsters. After looking through countless stack trace, I managed to fix it.

Iterator pattern

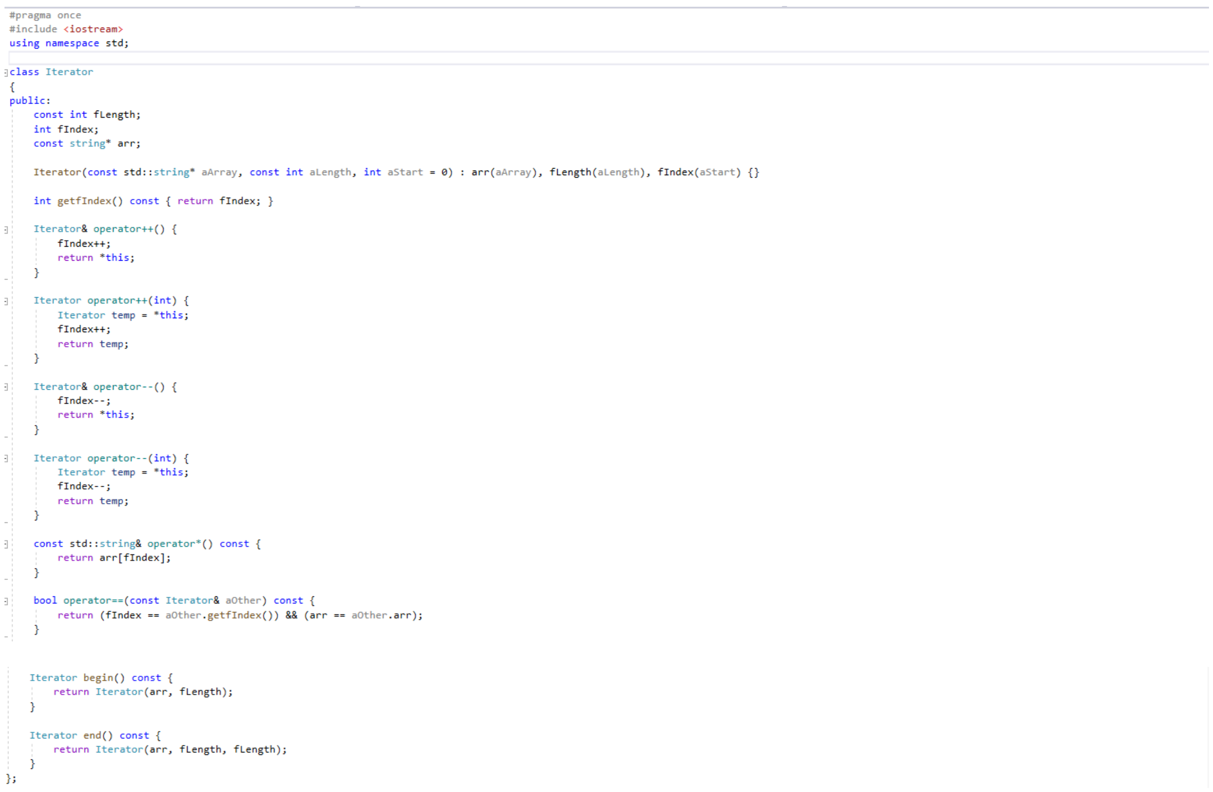
Task Description

The only way to get strong in this game is to gacha. The player can acquire gacha key from chest and use it on the gacha machine. For each key used, a lottery will be issued which contain item or attribute boost. Then, the player will throw dice to determine the number of slots/prizes could be unlocked from the lottery. Each lottery has 10 slots and I use an array to store it. However, I am thinking about to change from array to linked list in the future, but changes have to be made to client's code too.

Concept

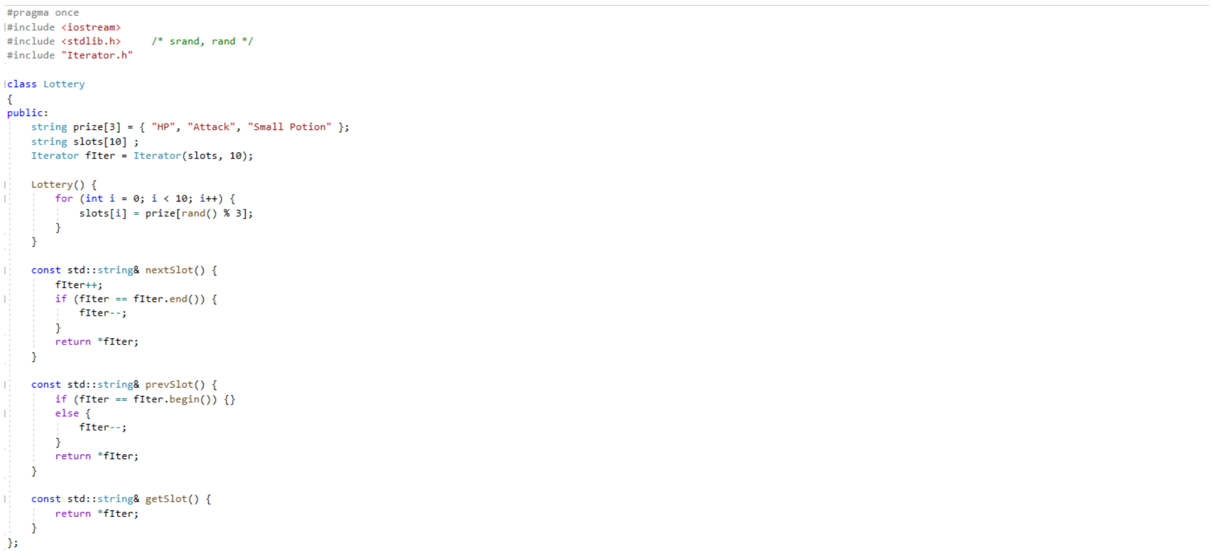
Iterator pattern could provide an abstraction and allow the client to use the same interface of the iterator without changes to the client's code. Iterator could provide encapsulation for the program too. Encapsulation is the idea that only necessary information is exposed to the client's code and hide all other implementation. Iterator allows the traverse of a sequence of data, and that will consume the current element after traverse to the next element, which means there is no way to go back to the previous element. A new iterator which begins at first element is needed if revisit an element is required. In my implementation, the lottery class will initialise the slots in the constructor, and pass the array into the iterator and will only interact with the iterator for the rest of the time.

Implementation & Output



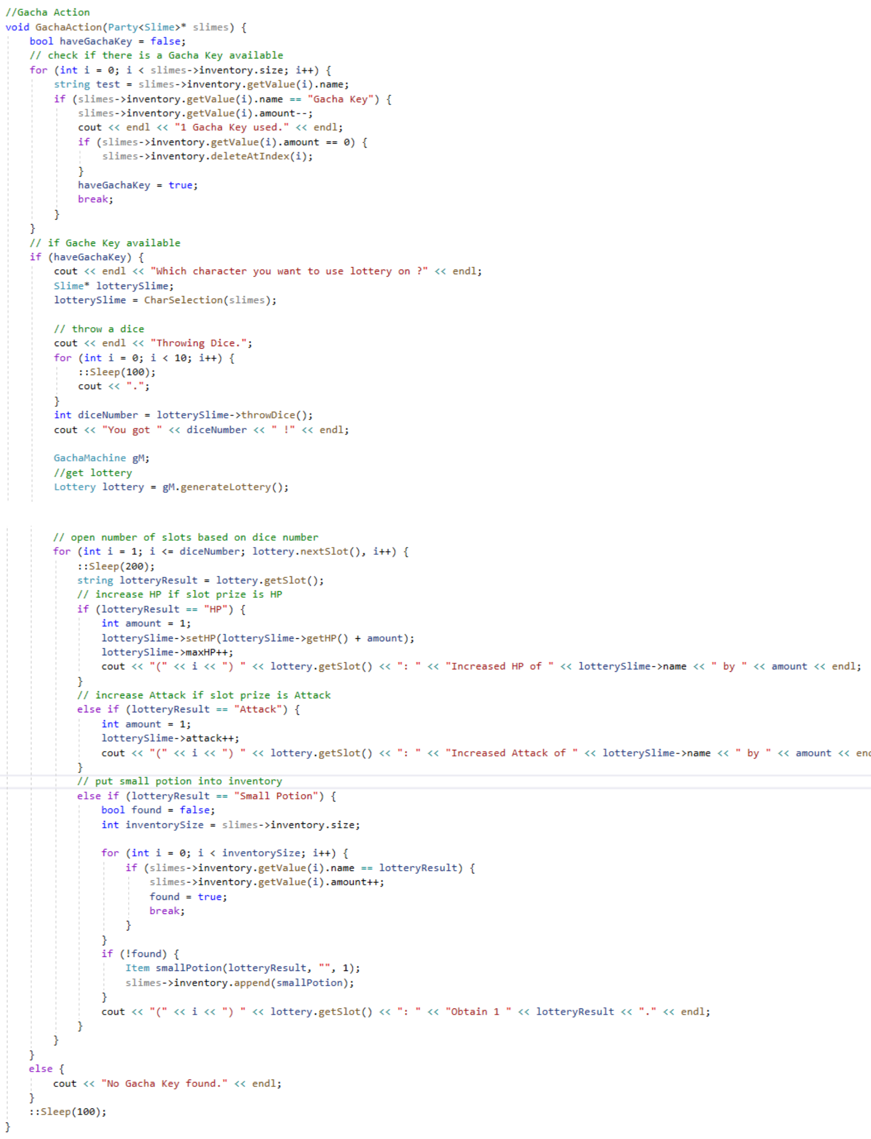
```cpp
#pragma once
#include <iostream>
using namespace std;

class Iterator
{
public:
    const int fLength;
    int fIndex;
    const string* arr;

    Iterator(const std::string* aArray, const int aLength, int aStart = 0) : arr(aArray), fLength(aLength), fIndex(aStart) {}

    int getfIndex() const { return fIndex; }

    Iterator& operator++() {
        fIndex++;
        return *this;
    }

    Iterator operator++(int) {
        Iterator temp = *this;
        fIndex++;
        return temp;
    }

    Iterator& operator--() {
        fIndex--;
        return *this;
    }

    Iterator operator--(int) {
        Iterator temp = *this;
        fIndex--;
        return temp;
    }

    const std::string& operator*() const {
        return arr[fIndex];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (arr == aOther.arr);
    }


    Iterator begin() const {
        return Iterator(arr, fLength);
    }

    Iterator end() const {
        return Iterator(arr, fLength, fLength);
    }
};
```

Figure: Iterator class (Iterator.h)



```cpp
#pragma once
#include <iostream>
#include <stdlib.h>     /* srand, rand */
#include "Iterator.h"

class Lottery
{
public:
    string prize[3] = { "HP", "Attack", "Small Potion" };
    string slots[10] ;
    Iterator fIter = Iterator(slots, 10);

    Lottery() {
        for (int i = 0; i < 10; i++) {
            slots[i] = prize[rand() % 3];
        }
    }

    const std::string& nextSlot() {
        fIter++;
        if (fIter == fIter.end()) {
            fIter--;
        }
        return *fIter;
    }

    const std::string& prevSlot() {
        if (fIter == fIter.begin()) {}
        else {
            fIter--;
        }
        return *fIter;
    }

    const std::string& getSlot() {
        return *fIter;
    }
};
```

Figure: Lottery class (Lottery.h)



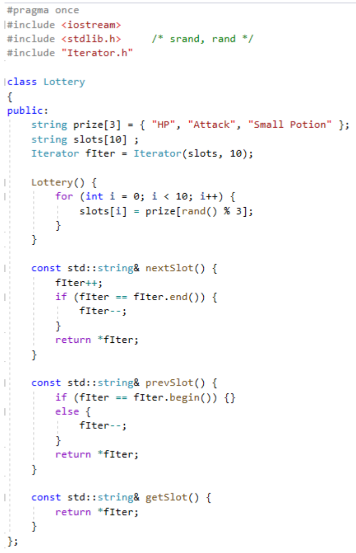
```cpp
//Gacha Action
void GachaAction(Party<Slime>* slimes) {
    bool haveGachaKey = false;
    // check if there is a Gacha Key available
    for (int i = 0; i < slimes->inventory.size; i++) {
        string test = slimes->inventory.getValue(i).name;
        if (slimes->inventory.getValue(i).name == "Gacha Key") {
            slimes->inventory.getValue(i).amount--;
            cout << endl << "1 Gacha Key used." << endl;
            if (slimes->inventory.getValue(i).amount == 0) {
                slimes->inventory.deleteAtIndex(i);
            }
            haveGachaKey = true;
            break;
        }
    }
    // if Gache Key available
    if (haveGachaKey) {
        cout << endl << "Which character you want to use lottery on ?" << endl;
        Slime* lotterySlime;
        lotterySlime = CharSelection(slimes);

        // throw a dice
        cout << endl << "Throwing Dice.";
        for (int i = 0; i < 10; i++) {
            ::Sleep(100);
            cout << ".";
        }
        int diceNumber = lotterySlime->throwDice();
        cout << "You got " << diceNumber << " !" << endl;

        GachaMachine gM;
        //get lottery
        Lottery lottery = gM.generateLottery();


        // open number of slots based on dice number
        for (int i = 1; i <= diceNumber; lottery.nextSlot(), i++) {
            ::Sleep(200);
            string lotteryResult = lottery.getSlot();
            // increase HP if slot prize is HP
            if (lotteryResult == "HP") {
                int amount = 1;
                lotterySlime->setHP(lotterySlime->getHP() + amount);
                lotterySlime->maxHP++;
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased HP of " << lotterySlime->name << " by " << amount << endl;
            }
            // increase Attack if slot prize is Attack
            else if (lotteryResult == "Attack") {
                int amount = 1;
                lotterySlime->attack++;
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased Attack of " << lotterySlime->name << " by " << amount << endl;
            }
            // put small potion into inventory
            else if (lotteryResult == "Small Potion") {
                bool found = false;
                int inventorySize = slimes->inventory.size;

                for (int i = 0; i < inventorySize; i++) {
                    if (slimes->inventory.getValue(i).name == lotteryResult) {
                        slimes->inventory.getValue(i).amount++;
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    Item smallPotion(lotteryResult, "", 1);
                    slimes->inventory.append(smallPotion);
                }
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Obtain 1 " << lotteryResult << "." << endl;
            }
        }
    }
    else {
        cout << "No Gacha Key found." << endl;
    }
    ::Sleep(100);
}
```

Figure: gacha in action (CodeReuse.h)

Troubleshooting
Not much problem had been encountered for this task as it is like what I had done in problem set 2.

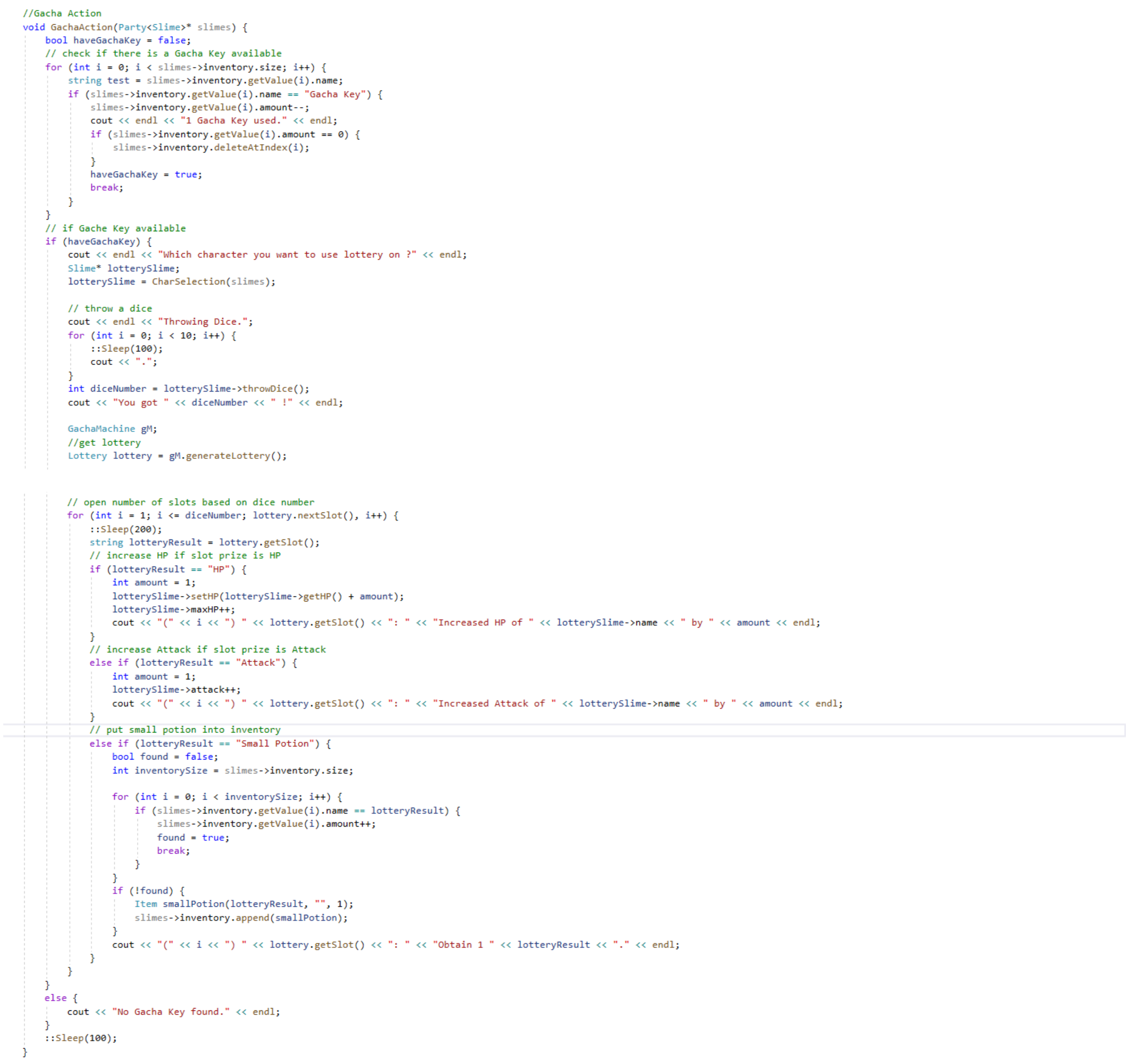
Adapter pattern
Task Description
With iterator pattern, changes in Lottery class is not needed even if linked list is used to replace array. However, changes still need to be made for the GachaAction() function in CodeReuse.h in the case of replacing Iterator pattern with other methods e.g. remote procedural call, inter-process communication. The code for using Iterator is a bit complicated too and this disobeys the encapsulation principle.

Concept
Another layer of abstraction could solve the issue. Adapter Pattern allows the interface of an existing class to be used without changes to the existing class's interface or client's code. In my implementation, the complex code for using iterator is confined in lottery class and the client only needs to use three simple functions, namely nextSlot(), prevSlot(), getSlot() without knowledge of the implementation. This is one of the main benefits. In the future, there may be a new class of Wishes that use REST API, but the client still could use the adapter interface of nextSlot() to fetch next wish.

Implementation & Output


```cpp
#pragma once
#include <iostream>
#include <stdlib.h>     /* srand, rand */
#include "Iterator.h"

class Lottery
{
public:
    string prize[3] = { "HP", "Attack", "Small Potion" };
    string slots[10] ;
    Iterator fIter = Iterator(slots, 10);

    Lottery() {
        for (int i = 0; i < 10; i++) {
            slots[i] = prize[rand() % 3];
        }
    }

    const std::string& nextSlot() {
        fIter++;
        if (fIter == fIter.end()) {
            fIter--;
        }
        return *fIter;
    }

    const std::string& prevSlot() {
        if (fIter == fIter.begin()) {}
        else {
            fIter--;
        }
        return *fIter;
    }

    const std::string& getSlot() {
        return *fIter;
    }
};
```

<u>Figure: Lottery class (Lottery.h)</u>



```cpp
//Gacha Action
void GachaAction(Party<Slime>* slimes) {
    bool haveGachaKey = false;
    // check if there is a Gacha Key available
    for (int i = 0; i < slimes->inventory.size; i++) {
        string test = slimes->inventory.getValue(i).name;
        if (slimes->inventory.getValue(i).name == "Gacha Key") {
            slimes->inventory.getValue(i).amount--;
            cout << endl << "1 Gacha Key used." << endl;
            if (slimes->inventory.getValue(i).amount == 0) {
                slimes->inventory.deleteAtIndex(i);
            }
            haveGachaKey = true;
            break;
        }
    }
    // if Gache Key available
    if (haveGachaKey) {
        cout << endl << "Which character you want to use lottery on ?" << endl;
        Slime* lotterySlime;
        lotterySlime = CharSelection(slimes);

        // throw a dice
        cout << endl << "Throwing Dice.";
        for (int i = 0; i < 10; i++) {
            ::Sleep(100);
            cout << ".";
        }
        int diceNumber = lotterySlime->throwDice();
        cout << "You got " << diceNumber << " !" << endl;

        GachaMachine gM;
        //get lottery
        Lottery lottery = gM.generateLottery();


        // open number of slots based on dice number
        for (int i = 1; i <= diceNumber; lottery.nextSlot(), i++) {
            ::Sleep(200);
            string lotteryResult = lottery.getSlot();
            // increase HP if slot prize is HP
            if (lotteryResult == "HP") {
                int amount = 1;
                lotterySlime->setHP(lotterySlime->getHP() + amount);
                lotterySlime->maxHP++;
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased HP of " << lotterySlime->name << " by " << amount << endl;
            }
            // increase Attack if slot prize is Attack
            else if (lotteryResult == "Attack") {
                int amount = 1;
                lotterySlime->attack++;
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased Attack of " << lotterySlime->name << " by " << amount << endl;
            }
            // put small potion into inventory
            else if (lotteryResult == "Small Potion") {
                bool found = false;
                int inventorySize = slimes->inventory.size;

                for (int i = 0; i < inventorySize; i++) {
                    if (slimes->inventory.getValue(i).name == lotteryResult) {
                        slimes->inventory.getValue(i).amount++;
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    Item smallPotion(lotteryResult, "", 1);
                    slimes->inventory.append(smallPotion);
                }
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Obtain 1 " << lotteryResult << "." << endl;
            }
        }
    }
    else {
        cout << "No Gacha Key found." << endl;
    }
    ::Sleep(100);
}
```

Figure: gacha in action (CodeReuse.h)

Troubleshooting

The initial difficulty is to understand what an adapter pattern is. Luckily, it is well described in the book head first design and design pattern: elements of reusable object-oriented software. The second difficulty is to apply it to the correct scenario. The purpose of design pattern is to reduce code complexity but apply it mindlessly will only lead the opposite result. In this case, I think my implementation is appropriate.

Builder pattern

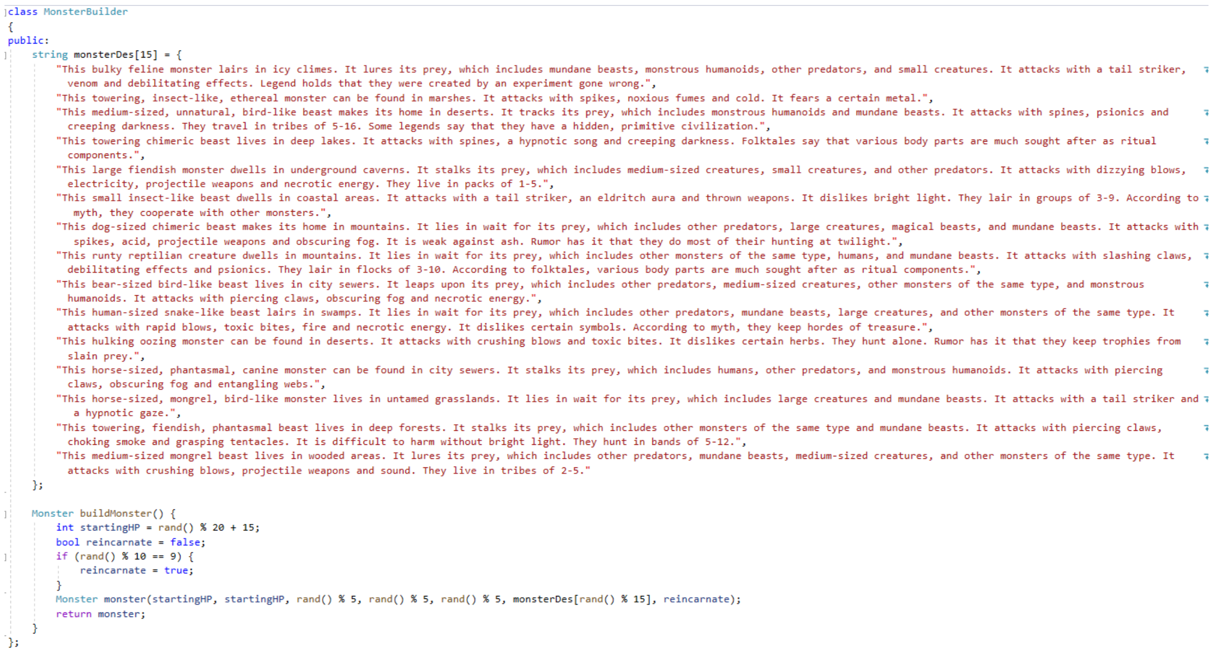
Task Description

Room is the building block of the world and monster is one of the elements in the room. This indicates that these elements may evolve into very complex class as the development progress to bring more content. Thus, the way of generating room and monster should fulfil the encapsulation and abstraction principles. I need to find a way to code it in the way that changes to room/monster generation will not affect the rest of the code.

Concept

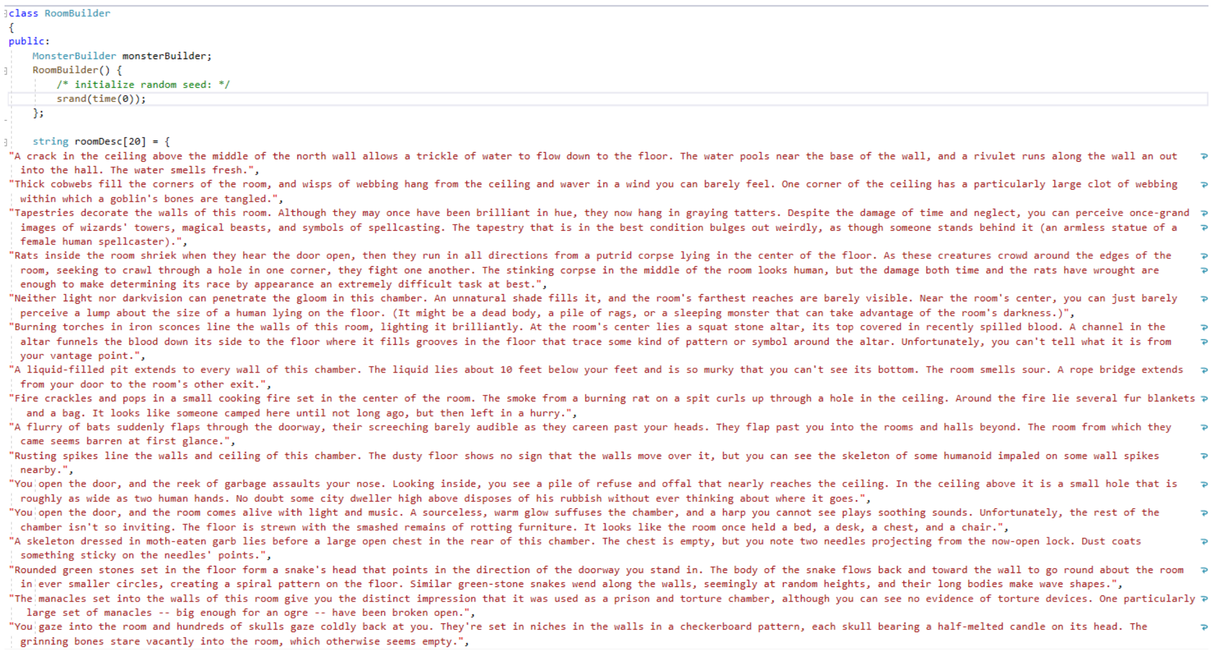
Builder pattern simplifies the creation of objects, instead of calling a complicated constructor or dozens of setter methods on the created object. An alternative way to generate object is through factory pattern, but it is more suitable for simple object. By letting the job of creating monster to the monster builder, any changes to the generation monster will be confined to the monster builder. This applies to room builder too. Builder pattern provides encapsulation by assign the task of creating object to a single class and abstraction by hiding the creation details from the client's code.

Implementation & Output

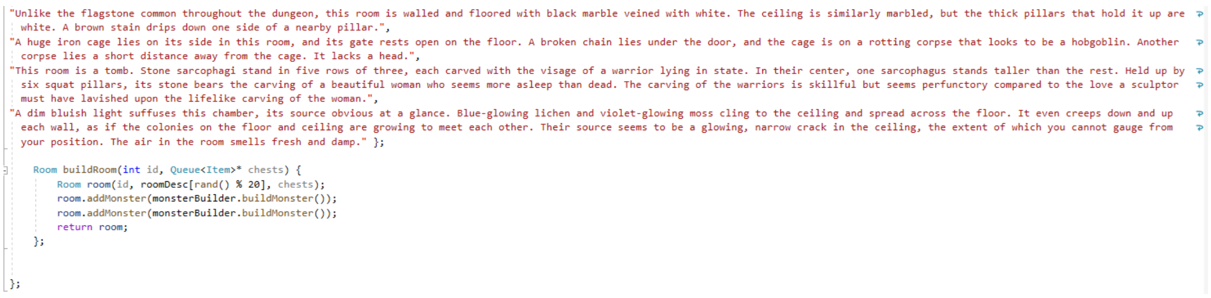


```
class MonsterBuilder
{
public:
    string monsterDes[15] = {
        "This bulky feline monster lairs in icy climes. It lures its prey, which includes mundane beasts, monstrous humanoids, other predators, and small creatures. It attacks with a tail striker,
         venom and debilitating effects. Legend holds that they were created by an experiment gone wrong.",
        "This towering, insect-like, ethereal monster can be found in marshes. It attacks with spikes, noxious fumes and cold. It fears a certain metal.",
        "This medium-sized, unnatural, bird-like beast makes its home in deserts. It tracks its prey, which includes monstrous humanoids and mundane beasts. It attacks with spines, psionics and
         creeping darkness. They travel in tribes of 5-16. Some legends say that they have a hidden, primitive civilization.",
        "This towering chimeric beast lives in deep lakes. It attacks with spines, a hypnotic song and creeping darkness. Folktales say that various body parts are much sought after as ritual
         components.",
        "This large fiendish monster dwells in underground caverns. It stalks its prey, which includes medium-sized creatures, small creatures, and other predators. It attacks with dizzying blows,
         electricity, projectile weapons and necrotic energy. They live in packs of 1-5.",
        "This small insect-like beast dwells in coastal areas. It attacks with a tail striker, an eldritch aura and thrown weapons. It dislikes bright light. They lair in groups of 3-9. According to
          myth, they cooperate with other monsters.",
        "This dog-sized chimeric beast makes its home in mountains. It lies in wait for its prey, which includes other predators, large creatures, magical beasts, and mundane beasts. It attacks with
          spikes, acid, projectile weapons and obscuring fog. It is weak against ash. Rumor has it that they do most of their hunting at twilight.",
        "This runty reptilian creature dwells in mountains. It lies in wait for its prey, which includes other monsters of the same type, humans, and mundane beasts. It attacks with slashing claws,
         debilitating effects and psionics. They lair in flocks of 3-10. According to folktales, various body parts are much sought after as ritual components.",
        "This bear-sized bird-like beast lives in city sewers. It leaps upon its prey, which includes other predators, medium-sized creatures, other monsters of the same type, and monstrous
         humanoids. It attacks with piercing claws, obscuring fog and necrotic energy.",
        "This human-sized snake-like beast lairs in swamps. It lies in wait for its prey, which includes other predators, mundane beasts, large creatures, and other monsters of the same type. It
         attacks with rapid blows, toxic bites, fire and necrotic energy. It dislikes certain symbols. According to myth, they keep hordes of treasure.",
        "This hulking oozing monster can be found in deserts. It attacks with crushing blows and toxic bites. It dislikes certain herbs. They hunt alone. Rumor has it that they keep trophies from
         slain prey.",
        "This horse-sized, phantasmal, canine monster can be found in city sewers. It stalks its prey, which includes humans, other predators, and monstrous humanoids. It attacks with piercing
         claws, obscuring fog and entangling webs.",
        "This horse-sized, mongrel, bird-like monster lives in untamed grasslands. It lies in wait for its prey, which includes large creatures and mundane beasts. It attacks with a tail striker and
          a hypnotic gaze.",
        "This towering, fiendish, phantasmal beast lives in deep forests. It stalks its prey, which includes other monsters of the same type and mundane beasts. It attacks with piercing claws,
         choking smoke and grasping tentacles. It is difficult to harm without bright light. They hunt in bands of 5-12.",
        "This medium-sized mongrel beast lives in wooded areas. It lures its prey, which includes other predators, mundane beasts, medium-sized creatures, and other monsters of the same type. It
         attacks with crushing blows, projectile weapons and sound. They live in tribes of 2-5."
    };

    Monster buildMonster() {
        int startingHP = rand() % 20 + 15;
        bool reincarnate = false;
        if (rand() % 10 == 9) {
            reincarnate = true;
        }
        Monster monster(startingHP, startingHP, rand() % 5, rand() % 5, rand() % 5, monsterDes[rand() % 15], reincarnate);
        return monster;
    }
};
```

Figure: the creation of monster is handled solely by monster builder (MonsterBuilder.h)

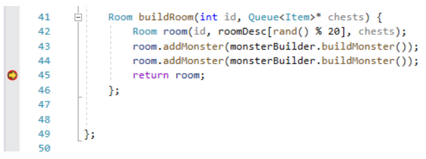


```
class RoomBuilder
{
public:
    MonsterBuilder monsterBuilder;
    RoomBuilder() {
        /* initialize random seed: */
        srand(time(0));
    };

    string roomDesc[20] = {
"A crack in the ceiling above the middle of the north wall allows a trickle of water to flow down to the floor. The water pools near the base of the wall, and a rivulet runs along the wall an out
 into the hall. The water smells fresh.",
"Thick cobwebs fill the corners of the room, and wisps of webbing hang from the ceiling and waver in a wind you can barely feel. One corner of the ceiling has a particularly large clot of webbing
 within which a goblin's bones are tangled.",
"Tapestries decorate the walls of this room. Although they may once have been brilliant in hue, they now hang in graying tatters. Despite the damage of time and neglect, you can perceive once-grand
 images of wizards' towers, magical beasts, and symbols of spellcasting. The tapestry that is in the best condition bulges out weirdly, as though someone stands behind it (an armless statue of a
 female human spellcaster).",
"Rats inside the room shriek when they hear the door open, then they run in all directions from a putrid corpse lying in the center of the floor. As these creatures crowd around the edges of the
 room, seeking to crawl through a hole in one corner, they fight one another. The stinking corpse in the middle of the room looks human, but the damage both time and the rats have wrought are
 enough to make determining its race by appearance an extremely difficult task at best.",
"Neither light nor darkvision can penetrate the gloom in this chamber. An unnatural shade fills it, and the room's farthest reaches are barely visible. Near the room's center, you can just barely
 perceive a lump about the size of a human lying on the floor. (It might be a dead body, a pile of rags, or a sleeping monster that can take advantage of the room's darkness.)",
"Burning torches in iron sconces line the walls of this room, lighting it brilliantly. At the room's center lies a squat stone altar, its top covered in recently spilled blood. A channel in the
 altar funnels the blood down its side to the floor where it fills grooves in the floor that trace some kind of pattern or symbol around the altar. Unfortunately, you can't tell what it is from
 your vantage point.",
"A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10 feet below your feet and is so murky that you can't see its bottom. The room smells sour. A rope bridge extends
 from your door to the room's other exit.",
"Fire crackles and pops in a small cooking fire set in the center of the room. The smoke from a burning rat on a spit curls up through a hole in the ceiling. Around the fire lie several fur blankets
  and a bag. It looks like someone camped here until not long ago, but then left in a hurry.",
"A flurry of bats suddenly flaps through the doorway, their screeching barely audible as they careen past your heads. They flap past you into the rooms and halls beyond. The room from which they
 came seems barren at first glance.",
"Rusting spikes line the walls and ceiling of this chamber. The dusty floor shows no sign that the walls move over it, but you can see the skeleton of some humanoid impaled on some wall spikes
 nearby.",
"You open the door, and the reek of garbage assaults your nose. Looking inside, you see a pile of refuse and offal that nearly reaches the ceiling. In the ceiling above it is a small hole that is
 roughly as wide as two human hands. No doubt some city dweller high above disposes of his rubbish without ever thinking about where it goes.",
"You open the door, and the room comes alive with light and music. A sourceless, warm glow suffuses the chamber, and a harp you cannot see plays soothing sounds. Unfortunately, the rest of the
 chamber isn't so inviting. The floor is strewn with the smashed remains of rotting furniture. It looks like the room once held a bed, a desk, a chest, and a chair.",
"A skeleton dressed in moth-eaten garb lies before a large open chest in the rear of this chamber. The chest is empty, but you note two needles projecting from the now-open lock. Dust coats
 something sticky on the needles' points.",
"Rounded green stones set in the floor form a snake's head that points in the direction of the doorway you stand in. The body of the snake flows back and toward the wall to go round about the room
 in ever smaller circles, creating a spiral pattern on the floor. Similar green-stone snakes wend along the walls, seemingly at random heights, and their long bodies make wave shapes.",
"The manacles set into the walls of this room give you the distinct impression that it was used as a prison and torture chamber, although you can see no evidence of torture devices. One particularly
  large set of manacles -- big enough for an ogre -- have been broken open.",
"You gaze into the room and hundreds of skulls gaze coldly back at you. They're set in niches in the walls in a checkerboard pattern, each skull bearing a half-melted candle on its head. The
 grinning bones stare vacantly into the room, which otherwise seems empty.",
```



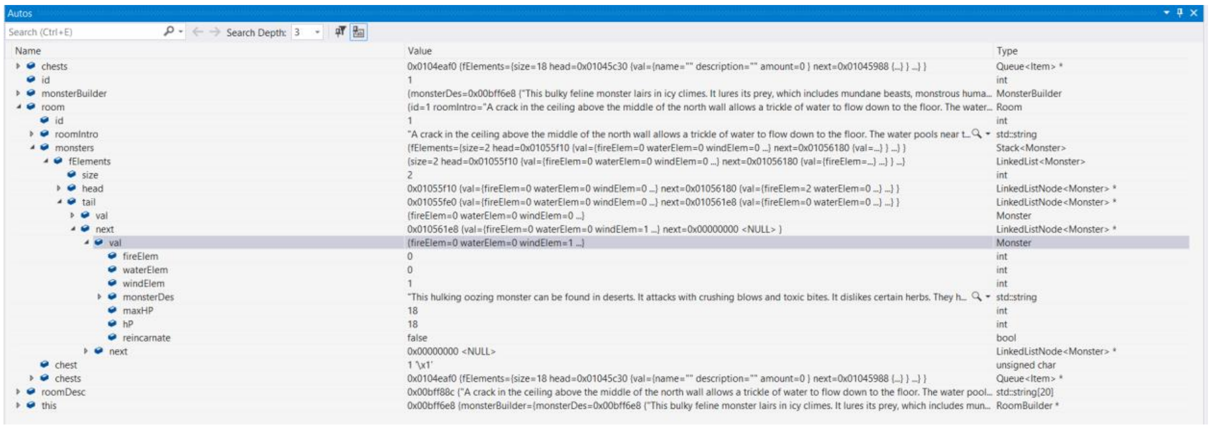
```
"Unlike the flagstone common throughout the dungeon, this room is walled and floored with black marble veined with white. The ceiling is similarly marbled, but the thick pillars that hold it up are
 white. A brown stain drips down one side of a nearby pillar.",
"A huge iron cage lies on its side in this room, and its gate rests open on the floor. A broken chain lies under the door, and the cage is on a rotting corpse that looks to be a hobgoblin. Another
 corpse lies a short distance away from the cage. It lacks a head.",
"This room is a tomb. Stone sarcophagi stand in five rows of three, each carved with the visage of a warrior lying in state. In their center, one sarcophagus stands taller than the rest. Held up by
 six squat pillars, its stone bears the carving of a beautiful woman who seems more asleep than dead. The carving of the warriors is skillful but seems perfunctory compared to the love a sculptor
 must have lavished upon the lifelike carving of the woman.",
"A dim bluish light suffuses this chamber, its source obvious at a glance. Blue-glowing lichen and violet-glowing moss cling to the ceiling and spread across the floor. It even creeps down and up
 each wall, as if the colonies on the floor and ceiling are growing to meet each other. Their source seems to be a glowing, narrow crack in the ceiling, the extent of which you cannot gauge from
 your position. The air in the room smells fresh and damp." };

    Room buildRoom(int id, Queue<Item>* chests) {
        Room room(id, roomDesc[rand() % 20], chests);
        room.addMonster(monsterBuilder.buildMonster());
        room.addMonster(monsterBuilder.buildMonster());
        return room;
    };


};
```

Figure: the creation of room is handled solely by room builder and the room builder is ignorant of monster creation detail (RoomBuilder.h)



```
    Room buildRoom(int id, Queue<Item>* chests) {
        Room room(id, roomDesc[rand() % 20], chests);
        room.addMonster(monsterBuilder.buildMonster());
        room.addMonster(monsterBuilder.buildMonster());
        return room;
    };


};
```

Figure: break point



```
Autos
Search (Ctrl+E)    Search Depth: 3

Name                      Value                                                                                                                                         Type
▸ chests                  0x0104eaf0 {fElements={size=18 head=0x01045c30 {val={name="" description="" amount=0 } next=0x01045988 {...} } ...} }                      Queue<Item> *
  id                      1                                                                                                                                             int
▸ monsterBuilder          {monsterDes=0x00bff6e8 {"This bulky feline monster lairs in icy climes. It lures its prey, which includes mundane beasts, monstrous huma...  MonsterBuilder
◂ room                    {id=1 roomIntro="A crack in the ceiling above the middle of the north wall allows a trickle of water to flow down to the floor. The water...  Room
    id                    1                                                                                                                                             int
  ▸ roomIntro             "A crack in the ceiling above the middle of the north wall allows a trickle of water to flow down to the floor. The water pools near t...  std::string
  ◂ monsters              {fElements={size=2 head=0x01055f10 {val={fireElem=0 waterElem=0 windElem=0 ...} next=0x01056180 {val=...} } ...} }                         Stack<Monster>
    ◂ fElements           {size=2 head=0x01055f10 {val={fireElem=0 waterElem=0 windElem=0 ...} next=0x01056180 {val={fireElem=...} ...} } ...}                      LinkedList<Monster>
        size              2                                                                                                                                             int
      ▸ head              0x01055f10 {val={fireElem=0 waterElem=0 windElem=0 ...} next=0x01056180 {val={fireElem=2 waterElem=0 ...} ...} }                           LinkedListNode<Monster> *
      ◂ tail              0x01055fe0 {val={fireElem=0 waterElem=0 windElem=0 ...} next=0x010561e8 {val={fireElem=0 waterElem=0 ...} ...} }                           LinkedListNode<Monster> *
        ▸ val             {fireElem=0 waterElem=0 windElem=0 ...}                                                                                                       Monster
        ◂ next            0x010561e8 {val={fireElem=0 waterElem=0 windElem=1 ...} next=0x00000000 <NULL> }                                                             LinkedListNode<Monster> *
          ◂ val           {fireElem=0 waterElem=0 windElem=1 ...}                                                                                                       Monster
              fireElem    0                                                                                                                                             int
              waterElem   0                                                                                                                                             int
              windElem    1                                                                                                                                             int
            ▸ monsterDes  "This hulking oozing monster can be found in deserts. It attacks with crushing blows and toxic bites. It dislikes certain herbs. They h...  std::string
              maxHP       18                                                                                                                                            int
              hP          18                                                                                                                                            int
              reincarnate false                                                                                                                                         bool
          ▸ next          0x00000000 <NULL>                                                                                                                             LinkedListNode<Monster> *
    chest                 1 '\x1'                                                                                                                                       unsigned char
  ▸ chests                0x0104eaf0 {fElements={size=18 head=0x01045c30 {val={name="" description="" amount=0 } next=0x01045988 {...} } ...} }                      Queue<Item> *
▸ roomDesc                0x00bff88c {"A crack in the ceiling above the middle of the north wall allows a trickle of water to flow down to the floor. The water pool... std::string[20]
▸ this                    0x00bff6e8 {monsterBuilder={monsterDes=0x00bff6e8 {"This bulky feline monster lairs in icy climes. It lures its prey, which includes mun...  RoomBuilder *
```

Figure: variable state at the break point

Troubleshooting

Builder pattern is easy to understand and the wrong application will not lead to catastrophic consequences. Thus, I had no trouble in solving this task.

Reference

CORMEN, T. H., & CORMEN, T. H. (2001). *Introduction to algorithms*. Cambridge, Mass, MIT Press.

Sedgewick, R. & Wayne, K. (2011), *Algorithms, 4th Edition.* , Addison-Wesley .

GAMMA, E., HELM, R., JOHNSON, R. E., & VLISSIDES, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, Mass, Addison-Wesley.

FREEMAN, E., ROBSON, E., SIERRA, K., & BATES, B. (2004). *Head First design patterns*. Sebastopol, CA, O'Reilly.

Appendix

CodeReuse.h

```
#pragma once

#include <iostream>

#include "Tree.h"

#include "Room.h"

#include "RoomBuilder.h"

#include "FireSlime.h"

#include "WindSlime.h"

#include "WaterSlime.h"

#include "LinkedList.h"

#include "Party.h"

#include "windows.h"

#include "GachaMachine.h"

#include <typeinfo>


class CodeReuse

{

public:

  void Overview(Party<Slime> slimes) {

    cout << endl << "Party Overview" << endl;

    slimes.listMembers();

    cout << endl << "Inventory Overview" << endl;

    slimes.inventory.display();

  }


  // Setup World

  TreeNode<DoublyLinkedList<Room>>* WorldSetup() {

    //pointer to queue of chests

    Queue<Item>* chests = new Queue<Item>();

    //get the location of buried Aztec Gold
    int aztecGoldLocation = rand() % 18;
    //fill in Item to the queue and Aztec Gold based on location
    for (int i = 0; i < 18; i++) {
        if (i == aztecGoldLocation) {
            Item aztecGold("Aztec Gold", "", 1);
            chests->enqueue(aztecGold);
        }
        else {
            Item gachaKey("Gacha Key", "", 1);
            chests->enqueue(gachaKey);
        }
    }
    RoomBuilder roomBuilder;
    // declare an array of 9 doubly linked list.
    // There will be 2 rooms in each doubly linked list.
    DoublyLinkedList<Room> rooms[9];
    //build room and insert into doubly linked list
    for (int i = 0; i < 9; i++) {
        rooms[i].append(roomBuilder.buildRoom(1, chests));
        rooms[i].append(roomBuilder.buildRoom(2, chests));
    }
    // put doubly linked list into array
    DoublyLinkedList<Room> arr[9] = { rooms[0], rooms[1], rooms[2], rooms[3], rooms[4], rooms[5],
rooms[6], rooms[7], rooms[8] };
    int n = sizeof(arr) / sizeof(arr[0]);
    // tree generator
    Tree<DoublyLinkedList<Room>> tree;
    // pointer to the root of tree
    TreeNode<DoublyLinkedList<Room>>* root = new TreeNode<DoublyLinkedList<Room>>();
    // build the tree
    root = tree.insertLevelOrder(arr, root, 0, n, NULL);
    return root;
}

// Setup Party
Party<Slime>* PartySetup() {
    Party<Slime>* slimes = new Party<Slime>();
    FireSlime* fireSlime = new FireSlime(80, 80, 5);
    WaterSlime* waterSlime = new WaterSlime(120, 120, 3);
    WindSlime* windSlime = new WindSlime(100, 100, 4);

    cout << "Who are YOU ?" << endl << endl;
    cout << "(1) Fire Slime..." << endl;
    cout << "(2) Water Slime..." << endl;
    cout << "(3) Wind Slime..." << endl;

    int charSelection;
    cout << "Input: ";
    cin >> charSelection;

    Slime* protagonistSlime = fireSlime;
    if (charSelection == 1) {
        protagonistSlime = fireSlime;
    }
    else if (charSelection == 2) {
        protagonistSlime = waterSlime;
    }
    else if (charSelection == 3) {
        protagonistSlime = windSlime;
    }
    cout << endl << protagonistSlime->name << " gain ";
    protagonistSlime->specialLuck();
    cout << endl << "Please tell me your name..." << endl;
    if (charSelection == 1) {
        cin >> fireSlime->name;
        system("cls");
        cout << fireSlime->name << "," << endl << "    " << " Water Slime and Wind Slime will be
joining you for this adventure..." << endl;
        slimes->addMember(*fireSlime);
        slimes->addMember(*waterSlime);
        slimes->addMember(*windSlime);
    }
    else if (charSelection == 2) {
        cin >> waterSlime->name;
        system("cls");
        cout << waterSlime->name << "," << endl << "    " << " Fire Slime and Wind Slime will be
joining you for this adventure..." << endl;
        slimes->addMember(*waterSlime);
        slimes->addMember(*fireSlime);
        slimes->addMember(*windSlime);
    }
    else if (charSelection == 3) {
        cin >> windSlime->name;
        system("cls");
        cout << windSlime->name << "," << endl << "    " << "Fire Slime and Water Slime will be joining
you for this adventure..." << endl;
        slimes->addMember(*windSlime);
        slimes->addMember(*waterSlime);
        slimes->addMember(*fireSlime);
    }

    Item slimeBanner("Slime Party Banner", "", 1);
    slimes->inventory.append(slimeBanner);
    Item smallPotion("Small Potion", "", 1);
    slimes->inventory.append(smallPotion);

    Item gachaKey("Gacha Key", "", 1);
    slimes->inventory.append(gachaKey);

    Overview(*slimes);

    cout << endl << "Let the Adventure Begins !!!" << endl;
    cout << endl << "Press any key to continue..." << endl;
    string placeHolder;
    cin >> placeHolder;

    return slimes;
}

// battle phase
boolean BattlePhase(Room* currRoom, Party<Slime>* slimes) {
    boolean fight = false;
    // if stack of monster is not empty
    while (!currRoom->monsters.isEmpty()) {
        fight = true;
        // get the top element as monster to fight
        Monster monster = currRoom->monsters.peek();

        cout << endl << "Monster appeared !!!" << endl;
        ::Sleep(100);
        cout << endl << monster.monsterDes << endl;

        ::Sleep(100);
    cout << endl << "============Battle Start============" << endl;
    ::Sleep(100);
    for (int j = 1; monster.hP >= 0; j++) {
        cout << endl << "Round " << j << endl;
        for (int i = 0; i < slimes->size(); i++) {
            ::Sleep(100);
            Slime& slime = slimes->getMembers(i);
            int damageReceived = 0;
            if (slime.type == "Fire Slime") {
                damageReceived = monster.fireElem;
            }
            else if (slime.type == "Water Slime") {
                damageReceived = monster.waterElem;
            }
            else if (slime.type == "Wind Slime") {
                damageReceived = monster.windElem;
            }

            if (slime.status != "Dead") {
                monster.hP -= slime.attack;
                cout << slime.name << " dealed " << slime.attack << " damage to Monster, received "
<< damageReceived << " damage." << endl;
                slime.setHP(slime.getHP() - damageReceived);
            }
        }
        if (slimes->getMembers(0).status == "Dead" && slimes->getMembers(1).status == "Dead"
&& slimes->getMembers(2).status == "Dead") {
            cout << endl << "All dead... all dead... all the slimes are dead..." << endl;
            cout << endl << "                              GAME OVER" << endl;
            exit(0);
        }
    }
```

```
        // battle finished, remove monster from stack by pop operation
        currRoom->monsters.pop();
        ::Sleep(100);
        cout << endl << "============Battle End============" << endl;

        ::Sleep(100);
        cout << endl << "Party Overview" << endl;
        slimes->listMembers();

        // if the popped monster reincarnate, then push a new reincarnated monster into the stack,
    and it will be the next monster to fight
        if (monster.reincarnate) {
            ::Sleep(100);
            cout << endl << "Monster reincarnate... Prepare for another battle..." << endl;
            MonsterBuilder monsterBuilder;
            currRoom->addMonster(monsterBuilder.buildMonster());
        }

        ::Sleep(100);
        // victory if there are no monsters left
        if (currRoom->monsters.isEmpty()) {
            cout << endl << "                              ";
            for (int i = 0; i < 3; i++) {
                ::Sleep(100);
                cout << " VICTORY";
            }
            cout << " !!!" << endl;
        }
        else {
            cout << endl << "Press any key to continue..." << endl;
            string wait;
            cin >> wait;
        }
        ::Sleep(100);
    }
    return fight;
}

//Character Selection
Slime* CharSelection(Party<Slime>* slimes) {
    slimes->listMembers();
    cout << "Input: ";
    int charSelection;
    cin >> charSelection;
    return &(slimes->getMembers(charSelection - 1));
}

//Use Item
void UseItem(Party<Slime>* slimes) {
    cout << endl << "List of Items: " << endl;
    slimes->inventory.display();
    cout << "Input: ";
    int itemUseSelection;
    cin >> itemUseSelection;

    if (slimes->inventory.getValue(itemUseSelection - 1).name == "Small Potion") {
        cout << endl << "Which character you want to use potion on ?" << endl;
        Slime* healSlime;
        healSlime = CharSelection(slimes);
        healSlime->setHP(healSlime->getHP() + 10);
        slimes->inventory.getValue(itemUseSelection - 1).amount--;
        cout << endl << "1 Small Potion used." << endl;
        if (slimes->inventory.getValue(itemUseSelection - 1).amount == 0) {
            slimes->inventory.deleteAtIndex(itemUseSelection - 1);
        }
        cout << endl << healSlime->name << " recovered 10 hp." << endl;
    }
    else if (slimes->inventory.getValue(itemUseSelection - 1).name == "Gacha Key") {
        GachaAction(slimes);
    }
}

//Gacha Action
void GachaAction(Party<Slime>* slimes) {
    bool haveGachaKey = false;
    // check if there is a Gacha Key available
    for (int i = 0; i < slimes->inventory.size; i++) {
        string test = slimes->inventory.getValue(i).name;
        if (slimes->inventory.getValue(i).name == "Gacha Key") {
            slimes->inventory.getValue(i).amount--;
            cout << endl << "1 Gacha Key used." << endl;
            if (slimes->inventory.getValue(i).amount == 0) {
                slimes->inventory.deleteAtIndex(i);
            }
            haveGachaKey = true;
            break;
        }
    }
    // if Gache Key available
    if (haveGachaKey) {
        cout << endl << "Which character you want to use lottery on ?" << endl;
        Slime* lotterySlime;
        lotterySlime = CharSelection(slimes);
        // throw a dice
        cout << endl << "Throwing Dice.";
        for (int i = 0; i < 10; i++) {
            ::Sleep(100);
            cout << ".";
        }
        int diceNumber = lotterySlime->throwDice();
        cout << "You got " << diceNumber << " !" << endl;

        GachaMachine gM;
        //get lottery
        Lottery lottery = gM.generateLottery();

        // open number of slots based on dice number
        for (int i = 1; i <= diceNumber; lottery.nextSlot(), i++) {
            ::Sleep(200);
            string lotteryResult = lottery.getSlot();
            // increase HP if slot prize is HP
            if (lotteryResult == "HP") {
                int amount = 1;
                lotterySlime->setHP(lotterySlime->getHP() + amount);
                lotterySlime->maxHP++;
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased HP of " <<
lotterySlime->name << " by " << amount << endl;
            }
            // increase Attack if slot prize is Attack
            else if (lotteryResult == "Attack") {
                int amount = 1;
                lotterySlime->attack++;
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Increased Attack of " <<
lotterySlime->name << " by " << amount << endl;
```

```
            }
            // put small potion into inventory
            else if (lotteryResult == "Small Potion") {
                bool found = false;
                int inventorySize = slimes->inventory.size;

                for (int i = 0; i < inventorySize; i++) {
                    if (slimes->inventory.getValue(i).name == lotteryResult) {
                        slimes->inventory.getValue(i).amount++;
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    Item smallPotion(lotteryResult, "", 1);
                    slimes->inventory.append(smallPotion);
                }
                cout << "(" << i << ") " << lottery.getSlot() << ": " << "Obtain 1 " << lotteryResult << "." <<
endl;
            }
        }
    }
    else {
        cout << "No Gacha Key found." << endl;
    }
    ::Sleep(100);
}

// Move Action
void MoveAction(TreeNode<DoublyLinkedList<Room>>** currNode,
DoublyLinkedNode<Room>** currRoom) {
    cout << endl << "Move direction available: " << endl;
    // if there is next room, print front
    if ((*currRoom)->next) {
      cout << "   W(Front)" << endl;
    }
    // if there is left child node and the current room is the first room in the current node, print left
    if ((*currNode)->left && (*currRoom)->data.id == 1) {
      cout << "   A(Left)" << endl;
    }
    // if there is parent node and the current room is the first room in the current node, print back
    if (!((*currNode)->parent == NULL && (*currRoom)->data.id == 1)) {
      cout << "   S(Back)" << endl;
    }
    // if there is right child node and the current room is the first room in the current node, print right
    if ((*currNode)->right && (*currRoom)->data.id == 1) {
      cout << "   D(Right)" << endl;
    }

    string movement;
    cout << "Input: ";
    cin >> movement;

    // if player decided to move back
    if (movement == "S") {
      // if current room is the first room in current node and there is parent node for current node, then move to parent node and set current room as first room in parent node.
      if ((*currRoom)->data.id == 1 && (*currNode)->parent) {
        *currNode = (*currNode)->parent;
        *currRoom = (*currNode)->data.getNode(0);
      }
      else {
        // otherwise move to previous room in doubly linked list
        *currRoom = (*currRoom)->prev;
      }
    }
    // if there is next room for current room in doubly linked list
    else if (movement == "W") {
      *currRoom = (*currRoom)->next;
    }
    // the player select move right
    else if (movement == "D") {
      // if there is right child node and the current room is first room in current node, then move to right child node and set current room as first room in parent node.
      if ((*currNode)->right && (*currRoom)->data.id == 1) {
        *currNode = (*currNode)->right;
        *currRoom = (*currNode)->data.getNode(0);
      }
    }
    // the player select move left
    else if (movement == "A") {
      // if there is left child node and the current room is first room in current node, then move to left child node and set current room as first room in parent node.
      if ((*currNode)->left && (*currRoom)->data.id == 1) {
        *currNode = (*currNode)->left;
        *currRoom = (*currNode)->data.getNode(0);
      }
    }
  }
};
```

DoublyLinkedList.h

```
#pragma once
#include <iostream>
#include "DoublyLinkedNode.h"

template<class DataType>
```

```cpp
class DoublyLinkedList
{

public:
    int size = 0;
    DoublyLinkedNode<DataType>* head = new DoublyLinkedNode<DataType>();
    DoublyLinkedNode<DataType>* tail = new DoublyLinkedNode<DataType>();

    DoublyLinkedList() {}

    // copy constructor
    DoublyLinkedList(const DoublyLinkedList& src) {
        DoublyLinkedNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->data);
        }
    }

    // assignment operator
    DoublyLinkedList& operator=(DoublyLinkedList src)
    {
        swap(head, src.head);
        swap(tail, src.tail);
        size = src.size;
        return *this;
    }

    // destructor
    ~DoublyLinkedList()
    {
        DoublyLinkedNode<DataType>* curr = head;
        while (curr != NULL) {
            DoublyLinkedNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }

    // add at the start
    void prepend(DataType val)
    {
        DoublyLinkedNode<DataType>* node = new DoublyLinkedNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
            node->prev = head;
        }
        else {
            node->prev = head;
            node->next = head->next;
            head->next = node;
        }
        size++;
    }

   // add at the end
    void append(DataType val)
    {
        DoublyLinkedNode<DataType>* node = new DoublyLinkedNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
            node->prev = head;
        }
        else {
            node->prev = tail->next;
            tail->next->next = node;
            tail->next = node;
        }
        size++;
    }

    /* Get the value of the index-th node in the linked list. If the index is invalid,
throw error. */
    DataType& get(int index) {
        if (index > size) throw "Index Invalid";
        DoublyLinkedNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    DoublyLinkedNode<DataType>* getNode(int index) {
        if (index > size) throw "Index Invalid";
        DoublyLinkedNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp;
    }

    //should be correct
    /** Add a node of value val before the index-th node in the linked list. If index
equals to the length of linked list, the node will be appended to the end of linked
list. If index is greater than the length, the node will not be inserted. */
    void addAtIndex(int index, DataType val) {
        if (index > size) return;
        DoublyLinkedNode<DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        DoublyLinkedNode<DataType>* node = new DoublyLinkedNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
            node->prev = head;
        }
        else {
            node->next = temp->next;
            temp->next = node;
            node->prev = temp;
        }
        size++;
    }

    //should be correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        DoublyLinkedNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        DoublyLinkedNode< DataType>* deletedNode = temp->next;
```

```
            temp->next = deletedNode->next;
            deletedNode->next->prev = temp;
            size--;
            delete deletedNode;
         }


    void display() {
        DoublyLinkedNode<DataType>* node = head;
        while (node != NULL)
        {
            cout << node->data << " ";
            node = node->next;
        }
    }
};
```

DoublyLinkedNode.h

```
#pragma once
#include <iostream>
using namespace std;

template<class DataType>
class DoublyLinkedNode
{
public:
    DataType data;
    DoublyLinkedNode<DataType>* next;
    DoublyLinkedNode<DataType>* prev;
    DoublyLinkedNode(DataType val) {
        this->data = val;
        next = NULL;
        prev = NULL;
    }

    DoublyLinkedNode() {
        next = NULL;
        prev = NULL;
    }
};
```

FireSlime.h

```
#pragma once
#include "Slime.h"
class FireSlime :
    public Slime
{
public:
    FireSlime(int MaxHP, int HP, int attack, string Name = "Fire Slime") :
Slime(MaxHP, HP, attack, "Fire Slime", Name)  {};

       void specialLuck() {
              attack = attack * 2;
              cout << "Enhanced Attack by 2x" << endl;
       }
};
```

GachaMachine.h

```
#pragma once
#include <iostream>
#include <stdlib.h>     /* srand, rand */
#include "Iterator.h"
#include "Lottery.h"

using namespace std;

class GachaMachine
{
public:
	GachaMachine() {

	}

	Lottery generateLottery() {
		return Lottery();
	}
};
```

Item.h

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

using namespace std;

class Item
{
public:
	string name;
	string description;
	int amount;

	Item() {};

	Item(string Name, string Description, int Amount) :
		name(Name), description(Description), amount(Amount) {};

	friend ostream& operator<<(ostream& aOstream, Item item) {
		aOstream << item.name << ": " << item.amount << endl;
		return aOstream;
	}
};
```

Iterator.h

```
#pragma once
#include <iostream>
using namespace std;
```

```
class Iterator
{
public:
        const int fLength;
        int fIndex;
        const string* arr;

    Iterator(const std::string* aArray, const int aLength, int aStart = 0) :
arr(aArray), fLength(aLength), fIndex(aStart) {}

        int getfIndex() const { return fIndex; }

    Iterator& operator++() {
        fIndex++;
        return *this;
    }

    Iterator operator++(int) {
        Iterator temp = *this;
        fIndex++;
        return temp;
    }

    Iterator& operator--() {
        fIndex--;
        return *this;
    }

    Iterator operator--(int) {
        Iterator temp = *this;
        fIndex--;
        return temp;
    }

    const std::string& operator*() const {
        return arr[fIndex];
    }

    bool operator==(const Iterator& aOther) const {
        return (fIndex == aOther.getfIndex()) && (arr == aOther.arr);
    }

    Iterator begin() const {
        return Iterator(arr, fLength);
    }

    Iterator end() const {
        return Iterator(arr, fLength, fLength);
    }
};
```

LinkedList.h

```
#pragma once
#include "LinkedListNode.h"
#include "windows.h"

template <class DataType>
```

```
class LinkedList
{
public:
    /** Initialize your data structure here. */
    int size = 0;
    LinkedListNode<DataType>* head = new LinkedListNode<DataType>();
    LinkedListNode<DataType>* tail = new LinkedListNode<DataType>();

    LinkedList() {}

    // copy constructor
    LinkedList(const LinkedList& src) {
        LinkedListNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->val);
        }
    }

    // assignment operator
    LinkedList& operator=(LinkedList src)
    {
        swap(head, src.head);
        size = src.size;
        return *this;
    }

    //destructor
    ~LinkedList()
    {
        LinkedListNode<DataType>* curr = head;
        while (curr != NULL) {
            LinkedListNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType& getValue(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType get(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    //correct
    /** Add a node of value val before the first element of the linked list. After the
insertion, the new node will be the first node of the linked list. */
    void prepend(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = head->next;
            head->next = node;
        }
        size++;
    }

    //correct
    /** Append a node of value val to the last element of the linked list. */
    void append(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            tail->next->next = node;
            tail->next = node;
        }
        size++;
    }

    void display() {
        LinkedListNode<DataType>* temp = head;
        int counter = 0;
        while (temp->next != NULL) {
            Sleep(100);
            counter++;
            temp = temp->next;
            //will call the overloaded operator of the data type.
            cout << "(" << counter << ")" << temp->val ;
        }
    }

    //should be correct
    /** Add a node of value val before the index-th node in the linked list. If index
equals to the length of linked list, the node will be appended to the end of linked
list. If index is greater than the length, the node will not be inserted. */
    void addAtIndex(int index, DataType val) {
        if (index > size) return;
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = temp->next;
            temp->next = node;
        }
        size++;
    }

    //correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        LinkedListNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode< DataType>* deletedNode = temp->next;
        temp->next = deletedNode->next;
        size--;
        delete deletedNode;
    }
};
```

LinkedListNode.h

```
#pragma once
#include "LinkedListNode.h"
#include "windows.h"

template <class DataType>
class LinkedList
{
public:
    /** Initialize your data structure here. */
    int size = 0;
    LinkedListNode<DataType>* head = new LinkedListNode<DataType>();
    LinkedListNode<DataType>* tail = new LinkedListNode<DataType>();

    LinkedList() {}

    // copy constructor
    LinkedList(const LinkedList& src) {
        LinkedListNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->val);
        }
    }

    // assignment operator
    LinkedList& operator=(LinkedList src)
    {
        swap(head, src.head);
        size = src.size;
        return *this;
    }

    //destructor
    ~LinkedList()
    {
        LinkedListNode<DataType>* curr = head;
        while (curr != NULL) {
            LinkedListNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType& getValue(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType get(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    //correct
    /** Add a node of value val before the first element of the linked list. After the
insertion, the new node will be the first node of the linked list. */
    void prepend(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = head->next;
            head->next = node;
        }
        size++;
    }

    //correct
    /** Append a node of value val to the last element of the linked list. */
    void append(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            tail->next->next = node;
            tail->next = node;
        }
        size++;
    }

    void display() {
        LinkedListNode<DataType>* temp = head;
        int counter = 0;
        while (temp->next != NULL) {
            Sleep(100);
            counter++;
            temp = temp->next;
            //will call the overloaded operator of the data type.
            cout << "(" << counter << ")" << temp->val ;
        }
    }
    //should be correct
    /** Add a node of value val before the index-th node in the linked list. If index
equals to the length of linked list, the node will be appended to the end of linked
list. If index is greater than the length, the node will not be inserted. */
    void addAtIndex(int index, DataType val) {
        if (index > size) return;
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = temp->next;
            temp->next = node;
        }
        size++;
    }

    //correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        LinkedListNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode< DataType>* deletedNode = temp->next;
        temp->next = deletedNode->next;
        size--;
        delete deletedNode;
    }
};
```

<u>Lottery.h</u>

```
#pragma once
#include "LinkedListNode.h"
#include "windows.h"

template <class DataType>
class LinkedList
{
public:
    /** Initialize your data structure here. */
    int size = 0;
    LinkedListNode<DataType>* head = new LinkedListNode<DataType>();
    LinkedListNode<DataType>* tail = new LinkedListNode<DataType>();

    LinkedList() {}

    // copy constructor
    LinkedList(const LinkedList& src) {
        LinkedListNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->val);
        }
    }

    // assignment operator
    LinkedList& operator=(LinkedList src)
    {
        swap(head, src.head);
        size = src.size;
        return *this;
    }

    //destructor
    ~LinkedList()
    {
        LinkedListNode<DataType>* curr = head;
        while (curr != NULL) {
            LinkedListNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType& getValue(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType get(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    //correct
    /** Add a node of value val before the first element of the linked list. After the
insertion, the new node will be the first node of the linked list. */
    void prepend(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = head->next;
            head->next = node;
        }
        size++;
    }

    //correct
    /** Append a node of value val to the last element of the linked list. */
    void append(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            tail->next->next = node;
            tail->next = node;
        }
        size++;
    }

    void display() {
        LinkedListNode<DataType>* temp = head;
        int counter = 0;
        while (temp->next != NULL) {
            Sleep(100);
            counter++;
            temp = temp->next;
            //will call the overloaded operator of the data type.
            cout << "(" << counter << ")" << temp->val ;
        }
    }

    //should be correct
    /** Add a node of value val before the index-th node in the linked list. If index
equals to the length of linked list, the node will be appended to the end of linked
list. If index is greater than the length, the node will not be inserted. */
    void addAtIndex(int index, DataType val) {
        if (index > size) return;
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = temp->next;
            temp->next = node;
        }
        size++;
    }

    //correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        LinkedListNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode< DataType>* deletedNode = temp->next;
        temp->next = deletedNode->next;
        size--;
        delete deletedNode;
    }
};
```

Monster.h

```
#pragma once
#include "LinkedListNode.h"
#include "windows.h"

template <class DataType>
class LinkedList
{
public:
    /** Initialize your data structure here. */
    int size = 0;
    LinkedListNode<DataType>* head = new LinkedListNode<DataType>();
    LinkedListNode<DataType>* tail = new LinkedListNode<DataType>();

    LinkedList() {}

    // copy constructor
    LinkedList(const LinkedList& src) {
        LinkedListNode<DataType>* node = src.head;
        while (node->next != NULL)
        {
            node = node->next;
            append(node->val);
        }
    }

    // assignment operator
    LinkedList& operator=(LinkedList src)
    {
        swap(head, src.head);
        size = src.size;
        return *this;
    }

    //destructor
    ~LinkedList()
    {
        LinkedListNode<DataType>* curr = head;
        while (curr != NULL) {
            LinkedListNode<DataType>* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType& getValue(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    /** Get the value of the index-th node in the linked list. If the index is
invalid, throw error. */
    DataType get(int index) {
        if (index >= size) throw "Index Invalid";
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i <= index; i++) temp = temp->next;
        return temp->val;
    }

    //correct
    /** Add a node of value val before the first element of the linked list. After the
insertion, the new node will be the first node of the linked list. */
    void prepend(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = head->next;
            head->next = node;
        }
        size++;
    }

    //correct
    /** Append a node of value val to the last element of the linked list. */
    void append(DataType val) {
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            tail->next->next = node;
            tail->next = node;
        }
        size++;
    }

    void display() {
        LinkedListNode<DataType>* temp = head;
        int counter = 0;
        while (temp->next != NULL) {
            Sleep(100);
            counter++;
            temp = temp->next;
            //will call the overloaded operator of the data type.
            cout << "(" << counter << ")" << temp->val ;
        }
    }

    //should be correct
    /** Add a node of value val before the index-th node in the linked list. If index
equals to the length of linked list, the node will be appended to the end of linked
list. If index is greater than the length, the node will not be inserted. */
    void addAtIndex(int index, DataType val) {
        if (index > size) return;
        LinkedListNode<DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode<DataType>* node = new LinkedListNode<DataType>(val);
        if (head->next == NULL && tail->next == NULL) {
            head->next = node;
            tail->next = node;
        }
        else {
            node->next = temp->next;
            temp->next = node;
        }
        size++;
    }

    //correct
    /** Delete the index-th node in the linked list, if the index is valid. */
    void deleteAtIndex(int index) {
        if (index >= size) return;
        LinkedListNode< DataType>* temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;
        }
        LinkedListNode< DataType>* deletedNode = temp->next;
        temp->next = deletedNode->next;
        size--;
        delete deletedNode;
    }
};
```

MonsterBuilder.h

```
#pragma once
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include "Monster.h"
#include <string>
#include <iostream>
using namespace std;

class MonsterBuilder
{
public:
       string monsterDes[15] = {
              "This bulky feline monster lairs in icy climes. It lures its prey, which
includes mundane beasts, monstrous humanoids, other predators, and small creatures. It
attacks with a tail striker, venom and debilitating effects. Legend holds that they
were created by an experiment gone wrong.",
              "This towering, insect-like, ethereal monster can be found in marshes.
It attacks with spikes, noxious fumes and cold. It fears a certain metal.",
              "This medium-sized, unnatural, bird-like beast makes its home in
deserts. It tracks its prey, which includes monstrous humanoids and mundane beasts. It
attacks with spines, psionics and creeping darkness. They travel in tribes of 5-16.
Some legends say that they have a hidden, primitive civilization.",
              "This towering chimeric beast lives in deep lakes. It attacks with
spines, a hypnotic song and creeping darkness. Folktales say that various body parts
are much sought after as ritual components.",
              "This large fiendish monster dwells in underground caverns. It stalks
its prey, which includes medium-sized creatures, small creatures, and other predators.
It attacks with dizzying blows, electricity, projectile weapons and necrotic energy.
They live in packs of 1-5.",
              "This small insect-like beast dwells in coastal areas. It attacks with a
tail striker, an eldritch aura and thrown weapons. It dislikes bright light. They lair
in groups of 3-9. According to myth, they cooperate with other monsters.",
		"This dog-sized chimeric beast makes its home in mountains. It lies in wait for its prey, which includes other predators, large creatures, magical beasts, and mundane beasts. It attacks with spikes, acid, projectile weapons and obscuring fog. It is weak against ash. Rumor has it that they do most of their hunting at twilight.",
		"This runty reptilian creature dwells in mountains. It lies in wait for its prey, which includes other monsters of the same type, humans, and mundane beasts. It attacks with slashing claws, debilitating effects and psionics. They lair in flocks of 3-10. According to folktales, various body parts are much sought after as ritual components.",
		"This bear-sized bird-like beast lives in city sewers. It leaps upon its prey, which includes other predators, medium-sized creatures, other monsters of the same type, and monstrous humanoids. It attacks with piercing claws, obscuring fog and necrotic energy.",
		"This human-sized snake-like beast lairs in swamps. It lies in wait for its prey, which includes other predators, mundane beasts, large creatures, and other monsters of the same type. It attacks with rapid blows, toxic bites, fire and necrotic energy. It dislikes certain symbols. According to myth, they keep hordes of treasure.",
		"This hulking oozing monster can be found in deserts. It attacks with crushing blows and toxic bites. It dislikes certain herbs. They hunt alone. Rumor has it that they keep trophies from slain prey.",
		"This horse-sized, phantasmal, canine monster can be found in city sewers. It stalks its prey, which includes humans, other predators, and monstrous humanoids. It attacks with piercing claws, obscuring fog and entangling webs.",
		"This horse-sized, mongrel, bird-like monster lives in untamed grasslands. It lies in wait for its prey, which includes large creatures and mundane beasts. It attacks with a tail striker and a hypnotic gaze.",
		"This towering, fiendish, phantasmal beast lives in deep forests. It stalks its prey, which includes other monsters of the same type and mundane beasts. It attacks with piercing claws, choking smoke and grasping tentacles. It is difficult to harm without bright light. They hunt in bands of 5-12.",
		"This medium-sized mongrel beast lives in wooded areas. It lures its prey, which includes other predators, mundane beasts, medium-sized creatures, and other monsters of the same type. It attacks with crushing blows, projectile weapons and sound. They live in tribes of 2-5."
	};

	Monster buildMonster() {
		int startingHP = rand() % 20 + 15;
		bool reincarnate = false;
		if (rand() % 10 == 9) {
			reincarnate = true;
		}
		Monster monster(startingHP, startingHP, rand() % 5, rand() % 5, rand() % 5, monsterDes[rand() % 15], reincarnate);
		return monster;
	}
};
```

Party.h

```
#pragma once
#include "Slime.h"

template <class DataType>
class Party
{
public:
	LinkedList<DataType> slimes;
	LinkedList<Item> inventory;

	Party() {}

	void addMember(DataType newMember) {
		slimes.append(newMember);
	}

	void listMembers() {
		slimes.display();
	}

	DataType& getMembers(int index) {
		return slimes.getValue(index);
	}

	int size() {
		return slimes.size;
	}
};
```

Queue.h

```
#pragma once
#include "LinkedList.h"

template<class DataType>
class Queue
{
private:
	LinkedList<DataType> fElements;

public:
	bool isEmpty() {
		return fElements.size == 0;
	}

	int size() {
		return fElements.size();
	}

	void enqueue(DataType val) {
		fElements.append(val);
	}

	DataType dequeue() {
		if (!isEmpty()) {
			DataType removed = fElements.get(0);
			fElements.deleteAtIndex(0);
			return removed;
		}
		else
			throw std::underflow_error("Queue is empty!");
	}
};
```

Room.h

```
#pragma once
#include "Stack.h"
#include "Monster.h"
#include "DoublyLinkedList.h"
#include "DoublyLinkedNode.h"
#include "LinkedList.h"
#include "Item.h"
#include "Queue.h"

class Room
{
public:
	int id;
	string roomIntro;
	Stack<Monster> monsters;
	boolean chest = true;
	Queue<Item>* chests;

	Room() {};

	Room(int ID, string RoomIntro, Queue<Item>* Chests): id(ID),
roomIntro(RoomIntro), chests(Chests) {
	};

	friend ostream& operator<<(ostream& aOstream, Room room) {
		aOstream << room.roomIntro << endl;
		return aOstream;
	}

	Item openChest() {
		chest = false;
		return chests->dequeue();
	}

	void addMonster(Monster monster) {
		monsters.push(monster);
	}

	~Room() {};
};
```

RoomBuilder.h

```
#pragma once
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include "Room.h"
#include "MonsterBuilder.h"
#include <string>
#include <iostream>
using namespace std;

class RoomBuilder
{
public:
	MonsterBuilder monsterBuilder;
	RoomBuilder() {
		/* initialize random seed: */
		srand(time(0));
	};

	string roomDesc[20] = {
"A crack in the ceiling above the middle of the north wall allows a trickle of water
to flow down to the floor. The water pools near the base of the wall, and a rivulet
runs along the wall an out into the hall. The water smells fresh.",
"Thick cobwebs fill the corners of the room, and wisps of webbing hang from the
ceiling and waver in a wind you can barely feel. One corner of the ceiling has a
particularly large clot of webbing within which a goblin's bones are tangled.",
"Tapestries decorate the walls of this room. Although they may once have been
brilliant in hue, they now hang in graying tatters. Despite the damage of time and
neglect, you can perceive once-grand images of wizards' towers, magical beasts, and
symbols of spellcasting. The tapestry that is in the best condition bulges out
weirdly, as though someone stands behind it (an armless statue of a female human
spellcaster).",
"Rats inside the room shriek when they hear the door open, then they run in all
directions from a putrid corpse lying in the center of the floor. As these creatures
crowd around the edges of the room, seeking to crawl through a hole in one corner,
they fight one another. The stinking corpse in the middle of the room looks human, but
the damage both time and the rats have wrought are enough to make determining its race
by appearance an extremely difficult task at best.",
"Neither light nor darkvision can penetrate the gloom in this chamber. An unnatural
shade fills it, and the room's farthest reaches are barely visible. Near the room's
center, you can just barely perceive a lump about the size of a human lying on the
floor. (It might be a dead body, a pile of rags, or a sleeping monster that can take
advantage of the room's darkness.)",
"Burning torches in iron sconces line the walls of this room, lighting it brilliantly.
At the room's center lies a squat stone altar, its top covered in recently spilled
blood. A channel in the altar funnels the blood down its side to the floor where it
fills grooves in the floor that trace some kind of pattern or symbol around the altar.
Unfortunately, you can't tell what it is from your vantage point.",
"A liquid-filled pit extends to every wall of this chamber. The liquid lies about 10
feet below your feet and is so murky that you can't see its bottom. The room smells
sour. A rope bridge extends from your door to the room's other exit.",
"Fire crackles and pops in a small cooking fire set in the center of the room. The
smoke from a burning rat on a spit curls up through a hole in the ceiling. Around the
fire lie several fur blankets and a bag. It looks like someone camped here until not
long ago, but then left in a hurry.",
"A flurry of bats suddenly flaps through the doorway, their screeching barely audible
as they careen past your heads. They flap past you into the rooms and halls beyond.
The room from which they came seems barren at first glance.",
"Rusting spikes line the walls and ceiling of this chamber. The dusty floor shows no
sign that the walls move over it, but you can see the skeleton of some humanoid
impaled on some wall spikes nearby.",
"You open the door, and the reek of garbage assaults your nose. Looking inside, you
see a pile of refuse and offal that nearly reaches the ceiling. In the ceiling above
it is a small hole that is roughly as wide as two human hands. No doubt some city
dweller high above disposes of his rubbish without ever thinking about where it
goes.",
"You open the door, and the room comes alive with light and music. A sourceless, warm
glow suffuses the chamber, and a harp you cannot see plays soothing sounds.
Unfortunately, the rest of the chamber isn't so inviting. The floor is strewn with the
smashed remains of rotting furniture. It looks like the room once held a bed, a desk,
a chest, and a chair.",
"A skeleton dressed in moth-eaten garb lies before a large open chest in the rear of
this chamber. The chest is empty, but you note two needles projecting from the now-
open lock. Dust coats something sticky on the needles' points.",
"Rounded green stones set in the floor form a snake's head that points in the
direction of the doorway you stand in. The body of the snake flows back and toward the
wall to go round about the room in ever smaller circles, creating a spiral pattern on
```

```
the floor. Similar green-stone snakes wend along the walls, seemingly at random
heights, and their long bodies make wave shapes.",
"The manacles set into the walls of this room give you the distinct impression that it
was used as a prison and torture chamber, although you can see no evidence of torture
devices. One particularly large set of manacles -- big enough for an ogre -- have been
broken open.",
"You gaze into the room and hundreds of skulls gaze coldly back at you. They're set in
niches in the walls in a checkerboard pattern, each skull bearing a half-melted candle
on its head. The grinning bones stare vacantly into the room, which otherwise seems
empty.",
"Unlike the flagstone common throughout the dungeon, this room is walled and floored
with black marble veined with white. The ceiling is similarly marbled, but the thick
pillars that hold it up are white. A brown stain drips down one side of a nearby
pillar.",
"A huge iron cage lies on its side in this room, and its gate rests open on the floor.
A broken chain lies under the door, and the cage is on a rotting corpse that looks to
be a hobgoblin. Another corpse lies a short distance away from the cage. It lacks a
head.",
"This room is a tomb. Stone sarcophagi stand in five rows of three, each carved with
the visage of a warrior lying in state. In their center, one sarcophagus stands taller
than the rest. Held up by six squat pillars, its stone bears the carving of a
beautiful woman who seems more asleep than dead. The carving of the warriors is
skillful but seems perfunctory compared to the love a sculptor must have lavished upon
the lifelike carving of the woman.",
"A dim bluish light suffuses this chamber, its source obvious at a glance. Blue-
glowing lichen and violet-glowing moss cling to the ceiling and spread across the
floor. It even creeps down and up each wall, as if the colonies on the floor and
ceiling are growing to meet each other. Their source seems to be a glowing, narrow
crack in the ceiling, the extent of which you cannot gauge from your position. The air
in the room smells fresh and damp." };

	Room buildRoom(int id, Queue<Item>* chests) {
		Room room(id, roomDesc[rand() % 20], chests);
		room.addMonster(monsterBuilder.buildMonster());
		room.addMonster(monsterBuilder.buildMonster());
		return room;
	};
};
```

Slime.h

```
#pragma once
#include <string>
#include <iostream>
#include "LinkedList.h"
#include "Item.h"
#include <stdlib.h>     /* srand, rand */

using namespace std;

class Slime
{
protected:
	int hP;

public:
	string name;
	int maxHP;
	int attack;
	string type;
	string status;

	Slime(int MaxHP, int HP, int Attack, string Type, string Name) :
		maxHP(MaxHP), hP(HP), attack(Attack), type(Type), name(Name) {
		status = "Normal";
	};
	Slime() {};


	void setHP(int HP) {
		hP = HP;
		if (hP < 0) {
			hP = 0;
		}
		if (hP > maxHP) {
			hP = maxHP;
		}
		if (hP == 0 && status == "Normal") {
			status = "Dead";
			cout << name << " is dead." << endl;
		}
	}

	int getHP() {
		return hP;
	}

	int throwDice() {
		return rand() % 10 + 1;
	}

	virtual void specialLuck() {
		return ;
	}

	friend ostream& operator<<(ostream& aOstream, Slime slime) {
		aOstream << slime.name << " -> " << "HP: " << slime.hP << "/" <<
slime.maxHP << " Attack: " << slime.attack << endl;
		return aOstream;
	}
};
```

Stack.h

```
#pragma once
#include "LinkedList.h"
#include <iostream>
using namespace std;

template <class DataType>
class Stack
{
public:
	LinkedList<DataType> fElements;
    Stack() {}
    // add data at the end
    void push(DataType data)
    {
        fElements.append(data);
    }

    // check is empty or not
    int isEmpty()
    {
        return fElements.size == 0;
    }

    // Utility function to return top element in a stack
    DataType& peek()
    {
        // Check for empty stack
        if (!isEmpty())
            return fElements.getValue(fElements.size - 1);
        else
            exit(1);
    }

    // Utility function to pop top element from the stack
    DataType pop()
    {
        if (!isEmpty()) {
            DataType removed = fElements.get(fElements.size - 1);
            fElements.deleteAtIndex(fElements.size - 1);
            return removed;
        }
        else
            throw std::underflow_error("Queue is empty!");
    }

    // Function to print all the elements of the stack
    void display()
    {
        fElements.display();
    }
};
```

Tree.h

```
#pragma once
#include "TreeNode.h"
#include <iostream>
using namespace std;

template <class DataType>
class Tree
{
public:

    // Function to insert nodes in level order
    TreeNode<DataType>* insertLevelOrder(DataType arr[], TreeNode<DataType>* root,
        int i, int n, TreeNode<DataType>* parent)
    {
        // Base case for recursion
        if (i < n)
        {
            // create a node
            TreeNode<DataType>* node = new TreeNode<DataType>();
            // initialise with data in arrat
            node->data = arr[i];
            node->left = node->right = NULL;
            root = node;

            // se this node to point to its parent
            root->parent = parent;

            // insert left child
            root->left = insertLevelOrder(arr,
                root->left, 2 * i + 1, n, root);

            // insert right child
            root->right = insertLevelOrder(arr,
                root->right, 2 * i + 2, n, root);
        }
        return root;
    }

    // Function to print tree nodes in
// InOrder fashion
    void inOrder(TreeNode<DataType>* root)
    {
        if (root != NULL)
        {
            cout << root->data << " ";
            inOrder(root->left);
            inOrder(root->right);
        }
    }
};
```

TreeNode.h

```
#pragma once
#include <stdio.h>

template <class DataType>
class TreeNode
{
public:
       DataType data;
       TreeNode* parent;
       TreeNode* left;
       TreeNode* right;

};
```

WaterSlime.h

```
#pragma once
#include "Slime.h"
class WaterSlime :
    public Slime
{
public:
    WaterSlime(int MaxHP, int HP, int attack, string Name = "Water Slime") :
Slime(MaxHP, HP, attack, "Water Slime", Name) {};

       void specialLuck() {
              hP += 40;
              maxHP += 40;
              cout << "Increased HP by 40" << endl;
       }
};
```

WindSLime.h

```
#pragma once
#include "Slime.h"
class WindSlime :
    public Slime
{
public:
    WindSlime(int MaxHP, int HP, int attack, string Name = "Wind Slime") :
Slime(MaxHP, HP, attack, "Wind Slime", Name) {};

       void specialLuck() {
              attack += 3;
              hP += 20;
              maxHP += 20;
              cout << "Increased Attack by 3 and HP by 20" << endl;
       }
};
```

Programming Project 1.cpp

```
// Programming Project 1.cpp : This file contains the 'main' function. Program
execution begins and ends there.
//

#include <iostream>
#include "CodeReuse.h"

using namespace std;

// Starting Point
int main()
{
    /* initialize random seed: */
    srand(time(0));
    CodeReuse codeReuse;
    // get pointer to current node
    TreeNode<DoublyLinkedList<Room>>* currentNode = codeReuse.WorldSetup();
    // a pointer to pointer and initialise it with the pointer to current node
    TreeNode<DoublyLinkedList<Room>>** currentNodePtr = &currentNode;
    // get pointer to the first room in current node
    DoublyLinkedNode<Room>* currentRoom = (*currentNodePtr)->data.getNode(0);
    // a pointer to pointer and initialise it with the pointer to current room
    DoublyLinkedNode<Room>** currentRoomPtr = &currentRoom;
    Party<Slime>* slimes = codeReuse.PartySetup();

    while (true) {
        if (currentNodePtr) {
            system("cls");
            // Entered room
            cout << "Entered room..." << endl;
            // print current room description
            cout << endl << (*currentRoomPtr)->data.roomIntro << endl;
            ::Sleep(100);

            // Battle if there is any monster
            boolean fight = codeReuse.BattlePhase(&((*currentRoomPtr)->data), slimes);

            while (true) {
                cout << endl << "Resting";
                for (int i = 0; i < 10; i++) {
                    ::Sleep(100);
                    cout << ".";
                }
                cout << endl;
                codeReuse.Overview(*slimes);
                cout << endl << "Action Available:" << endl;
                cout << "(1) Check Surrounding" << endl;
                cout << "(2) Use Item" << endl;
                cout << "(3) Gacha!!!" << endl;
                cout << "(4) Move..." << endl;

                string chooice;
                cout << "Input: ";
                cin >> chooice;

                // if check surrounding
                if (chooice == "1") {
                    if ((*currentRoomPtr)->data.chest
&& !((*currentRoomPtr)->data.chests->isEmpty())) {
                        Item chestItem = (*currentRoomPtr)->data.openChest();
                        if (chestItem.name == "Aztec Gold") {
                            cout << endl << "You had found Aztec Gold... You will be
cursed for a thousand years..." << endl;
                            cout << endl << "                                GAME
OVER";

                            exit(0);
                        }

                        bool found = false;
                        for (int i = 0; i < slimes->inventory.size; i++) {
                            if (slimes->inventory.getValue(i).name == chestItem.name)
{

                                slimes->inventory.getValue(i).amount++;
                                found = true;
                                break;
                            }
                        }
                        if (!found) {
                            Item smallPotion(chestItem.name, "", 1);
                            slimes->inventory.append(smallPotion);
                        }
                        cout << endl << "Obtain 1 " << chestItem.name << "." << endl;
                    }
                    else {
                        cout << endl << "Nothing found..." << endl;
                    }
                }
                else if (chooice == "2") {
                    codeReuse.UseItem(slimes);
                }
                else if (chooice == "3") {
                    codeReuse.GachaAction(slimes);
                }
                // move action
                else if (chooice == "4") {
                    codeReuse.MoveAction(currentNodePtr, currentRoomPtr);
                    break;
                }
            }
        }
    }
}
```