Lab 4

ELEC 3150 – Object Oriented Programming (Fall 2023)

Nick Cebula

HW Q1: main.cpp (Chose to use default class constructor balance=500 at start)

```cpp
#include <iostream>
#include <string>
#include "Account.h"
using std::cin;
using std::cout;
using std::string;
using std::endl;

int main() {
    // Create an Account object on the heap
    Account* ptr_Account = new Account();

    // Check balance
    cout << "Balance: $" << ptr_Account->checkbalance() << endl;

    // Deposit $1000
    ptr_Account->deposit(1000);
    // Check balance
    cout << "Balance: $" << ptr_Account->checkbalance() << endl;

    // Withdraw $300
    ptr_Account->withdraw(300);
    // Check balance
    cout << "Balance: $" << ptr_Account->checkbalance() << endl;

    // Delete the object
    delete ptr_Account;
    return 0;
}
```

Account.h:

```cpp
#pragma once
#include <iostream>
#include <string>
using std::string;
class Account
{
    //attributes
    string name;
    int balance;
public:
    //constructor
    Account();
    Account(string in_name, int in_balance);
    //method
    void deposit(int in_money);
    void withdraw(int out_money);
    int checkbalance();
    string showname();
    //destructor
    ~Account();
};
```

Account.cpp

```cpp
#include "Account.h"

//constructor
Account::Account() {
    name = "NA";
    balance = 500;
}
Account::Account(string in_name, int in_balance) {
    in_name = name;
    in_balance = balance;
}
//method
void Account::deposit(int in_money) {
    balance += in_money;
}
void Account::withdraw(int out_money) {
    balance -= out_money;
}
int Account::checkbalance() {
    return balance;
}
string Account::showname() {
    return name;
}
//destructor
Account::~Account() {

}
```

Q2:

Main.cpp:

```cpp
Bank_Admin admin;
admin.update_name(*ptr_Account,"Peter");
cout << "Updated Account name: " << ptr_Account->showname() << endl;

// Add $10 interest to the account
admin.interest(*ptr_Account, 10);
cout << "Updated Account balance after interest: $" << ptr_Account->checkbalance() << endl;

// Change total balance of the account to $50
admin.update_balance(*ptr_Account, 50);
cout << "Updated Account balance: $" << ptr_Account->checkbalance() << endl;

// Delete the object
delete ptr_Account;
return 0;
```

Bank_Admin.h:

```cpp
#pragma once
#include <iostream>
#include <string>
using std::string;
#include "Account.h"


class Bank_Admin
{
private:
    //attributes
    string name;
    int balance;
public:
    //constructors

    //methods
    void update_name(Account& user, string in_name);
    void interest(Account& user, int interest_rate);
    void update_balance(Account& user, int in_balance);

    //destructor
    ~Bank_Admin();
};
```

Bank_Admin.cpp

```cpp
#include "Bank_Admin.h"

void Bank_Admin::update_name(Account& user, string in_name){
    user.name = in_name;
}

void Bank_Admin::interest(Account& user, int interest_rate){
    user.deposit(interest_rate);
}

void Bank_Admin::update_balance(Account& user, int in_balance){
    user.balance = in_balance;
}

Bank_Admin::~Bank_Admin(){
}
```

Results (q1&2 together):

```
Balance: $500
Balance: $1500
Balance: $1200
Updated Account name: Peter
Updated Account balance after interest: $1210
Updated Account balance: $50
```

Q1: Boxing Game

Assumptions:

Create a player and fight against computer
- Both player and computer have the following attribute
- username
- health: 100
- power: 100
- Both player and computer can do the following
- Jab
- Cross
- Guard
- At each turn, both player and computer must choose their option.
- Computer selects its option randomly
- Their health and power will be calculated based on their selection

Attack (impact to their power):
- Jab: Power is decreased by 10
- Cross: Power is decreased by 20
- Hook: Power is decreased by 30
- Uppercut: Power is decreased by 40
- Guard: check next slide

- Damage to their opponent:
  - Jab: Health is decreased by 10
  - Cross: Health is decreased by 20
  - Hook: Health is decreased by 30
  - Uppercut: Health is decreased by 40
  - Guard: Check next slide

	Attack	Damage (to opponent)
Jab	Their Power is decreased by 10	Opponent's health is decreased by 10
Cross	Their Power is decreased by 20	Opponent's health is decreased by 20
Hook	Their Power is decreased by 30	Opponent's health is decreased by 30
Uppercut	Their Power is decreased by 40	Opponent's health is decreased by 40

When a fighter chooses to guard:

Opponent's move	Jab	Cross	Hook	Uppercut	Guard
User's health	0	-5	-10	-20	+10
User's power	+25	+15	+10	0	+30

Main.cpp:

```cpp
int main() {
    int player_choice;
    int computersmove;
    srand((time(0)));
    cout << "Welcome to Boxing Game!" << endl;
    string player_name;
    cout << "Enter your name: ";
    cin >> player_name;

    Player player(player_name, 100, 10); //user1 is player
    Player computer("Bot",100, 100);  //user2 is computer

    while (player.show_health() > 0 && computer.show_health() > 0) {
        cout << endl;
        cout << "Your Turn:" << endl;
        cout << "1. Jab -10" << endl << "2. Cross -20" << endl << "3. Hook -30" << endl << "4. Uppercut -40" << endl << "5. Guard" << endl;
        if (player.show_power() < 0) {
            cout << "Your power is below 0. Your turn is skipped, and your power will increase by 20 in the next round." << endl;
            player.set_power(20);
        }
```

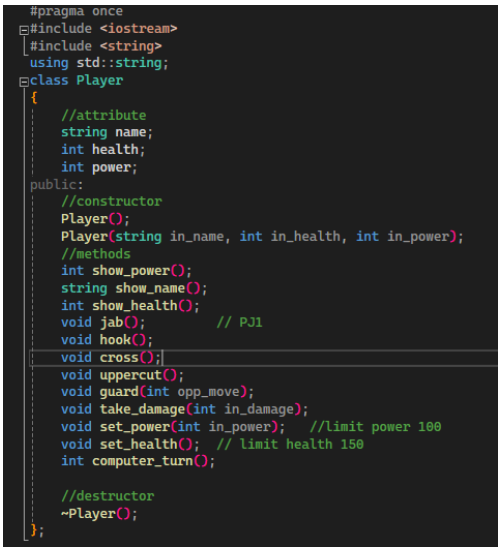
```
else {
    cout << "Enter your choice (1-5): ";
    cin >> player_choice;
    switch (player_choice) {
    case 1:
        player.jab();
        computer.take_damage(10);
        break;
    case 2:
        player.cross();
        computer.take_damage(20);
        break;
    case 3:
        player.hook();
        computer.take_damage(30);
        break;
    case 4:
        player.uppercut();
        computer.take_damage(30);
        break;
    case 5:
        player.guard(computersmove);
        break;
    default:
        cout << "Invalid choice. Please choose again." << endl;
        continue;
    }
    if (player.show_power() > 100) {
        player.set_power(100);
    }
    if (player.show_health() > 150) {
        player.set_health();
    }
}
```

```
if (computer.show_power() < 0) {
    cout << "Your power is below 0. Your turn is skipped, and your power will increase by 20 in the next round." << endl;
    player.set_power(20);
}
else {
    computersmove = computer.computer_turn();
    // Computer's turn
    switch (computersmove) {
    case 1:
        if (computersmove == 1) {
            cout << computer.show_name() << " Jab" << endl;
        }
        computer.jab();
        player.take_damage(10);
        break;
    case 2:
        if (computersmove == 2) {
            cout << computer.show_name() << " Hook" << endl;
        }
        computer.cross();
        player.take_damage(20);
        break;
    case 3:
        if (computersmove == 3) {
            cout << computer.show_name() << " Cross" << endl;
        }
        computer.hook();
        player.take_damage(30);
        break;
```

```
        case 4:
            if (computersmove == 4) {
                cout << computer.show_name() << " Uppercut" << endl;
            }
            computer.uppercut();
            player.take_damage(40);
            break;
        case 5:
            if (computersmove == 5) {
                cout << computer.show_name() << " Guard" << endl;
            }
            computer.guard(player_choice);
            break;
        default:
            cout << "Invalid choice. Please choose again." << endl;
            continue;
        }
        if (computer.show_power() > 100) {
            computer.set_power(100);
        }
        if (computer.show_health() > 150) {
            computer.set_health();
        }
        cout << computer.show_name() << " (Health: " << computer.show_health() << ")" << endl;
        cout << player.show_name() << " (Health: " << player.show_health() << ", Power: " << player.show_power() << ")" << endl;
    }
}
if (player.show_health() <= 0) {
    cout << "Game over! You lose." << endl;
}
else {
    cout << "Congratulations! You win." << endl;
}


return 0;
```

Explanation: In the main, the code starts off by making variables players choice and computers move and initializing random number generator. It asks the user to enter their name and creates a player and computer under Player class. A while loop controls the whole game under the conditions of player and computers health being greater than 0. There is a if loop for the condition of the player having power below 0, which will skip their turn and power increases by 20 next round. Then next else loop controls the players choice of their action, this will inflict damage upon the computer. The same if and else loop are applied for the application of the computers turn to do damage to the player which is done randomly. At the end there is an if and else condition for the game to end when either player or computer wins.

Player.h:

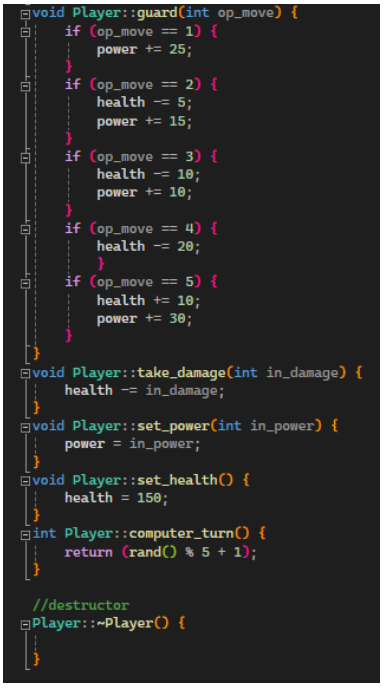


```
#pragma once
#include <iostream>
#include <string>
using std::string;
class Player
{
    //attribute
    string name;
    int health;
    int power;
public:
    //constructor
    Player();
    Player(string in_name, int in_health, int in_power);
    //methods
    int show_power();
    string show_name();
    int show_health();
    void jab();          // PJ1
    void hook();
    void cross();
    void uppercut();
    void guard(int opp_move);
    void take_damage(int in_damage);
    void set_power(int in_power);   //limit power 100
    void set_health();  // limit health 150
    int computer_turn();

    //destructor
    ~Player();
};
```

Explanation: In the player header it creates the class with attributes of name, health, and power. The constructor takes input of name, health, and power. Its methods are showing its attributes, it has abilities to inflict damage by jab, hook, cross, and uppercut. Then set power and health to change these values.

Player.cpp:

```
#include "Player.h"


//constructor
Player::Player() {
    name = "Default";
    health = 100;
    power = 100;
}
Player::Player(string in_name, int in_health, int in_power) {
    name = in_name;
    health = in_health;
    power = in_power;
}
//methods
int Player::show_power() {
    return power;
}
string Player::show_name() {
    return name;
}
int Player::show_health() {
    return health;
}
void Player::jab() {
    power -= 10;

}
void Player::cross() {
    power -= 20;

}
void Player::hook() {
    power -= 30;

}
void Player::uppercut() {
    power -= 40;

}
```



```
void Player::guard(int op_move) {
    if (op_move == 1) {
        power += 25;
    }
    if (op_move == 2) {
        health -= 5;
        power += 15;
    }
    if (op_move == 3) {
        health -= 10;
        power += 10;
    }
    if (op_move == 4) {
        health -= 20;
        }
    if (op_move == 5) {
        health += 10;
        power += 30;
    }
}
void Player::take_damage(int in_damage) {
    health -= in_damage;
}
void Player::set_power(int in_power) {
    power = in_power;
}
void Player::set_health() {
    health = 150;
}
int Player::computer_turn() {
    return (rand() % 5 + 1);
}

//destructor
Player::~Player() {

}
```

Explanation: In the player.cpp it defines the use created in the header file. Sets player default attributes, or has user input their traits with input constructor. Methods are determined by showing attributes will have an output of either string or int and return the number or word. The attack methods decrease power when used. For computers turn I have it as a method which is incorrect, I should have this in the main as a function to have player and computers turn to simplify.

Results:

```
Welcome to Boxing Game!
Enter your name: Nick

Your Turn:
1. Jab -10
2. Cross -20
3. Hook -30
4. Uppercut -40
5. Guard
Enter your choice (1-5): 3
Bot Guard
Bot (Health: 60)
Nick (Health: 100, Power: 70)

Your Turn:
1. Jab -10
2. Cross -20
3. Hook -30
4. Uppercut -40
5. Guard
Enter your choice (1-5): 2
Bot Jab
Bot (Health: 40)
Nick (Health: 90, Power: 50)
```

```
Your Turn:
1. Jab -10
2. Cross -20
3. Hook -30
4. Uppercut -40
5. Guard
Enter your choice (1-5): 1
Bot Cross
Bot (Health: 30)
Nick (Health: 60, Power: 40)

Your Turn:
1. Jab -10
2. Cross -20
3. Hook -30
4. Uppercut -40
5. Guard
Enter your choice (1-5): 5
Bot Cross
Bot (Health: 30)
Nick (Health: 20, Power: 50)
```

Question 2: Zombie Game

Assumptions:

Human (Attribute: name, health, bullet)

- Health (Max is 100)
- Method: Shoot, hit
- Bullet (Maximum is 7). If Bullet is empty, user can hit zombie and Zombie health will going down by 10.

Zombie (Attribute: name, health, weak point) – Create these object on the ***heap***

- Weak point is random for each zombie. It could be head or heart.
  - If user ***shoot*** at the right position, Zombie health is going to down by 50, otherwise it's going down by 25.
- Method: Bite (User has 30% chance of get bit by zombie. If he/she got bite, his/her health will be going down by 10 health of user.

Main.cpp:

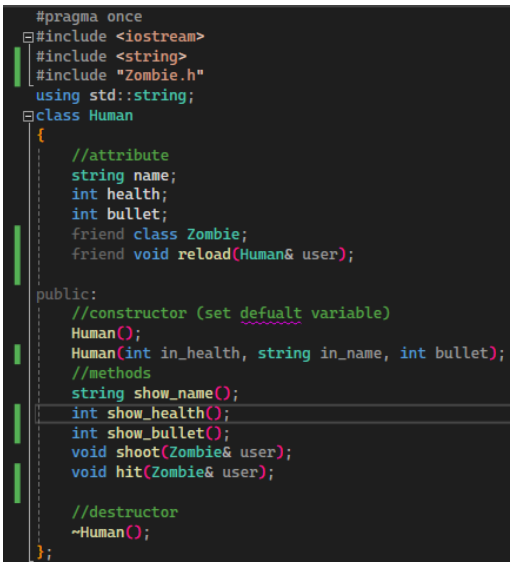
```cpp
#include <iostream>
#include <string>
#include "Human.h"
#include "Zombie.h"
#include <cstdlib>
using std::cin;
using std::cout;
using std::string;
using std::endl;

void heal(int turn, Human& human, Zombie& zom) {
    if (turn % 5 == 0) {
        if (rand() % 2 == 0) {
            // Create a new Human with health 100
            //Human user1();
        }
    }
}
void reload(Human& user) {
    user.bullet += 5;
    if (user.bullet > 7) {
        user.bullet = 7;
    }
}

int main() {
    Human player(100, "Human", 7);
    // Create 10 Zombies
    Zombie* zom[10];
    for (int i = 0; i < 10; i++) {
        zom[i] = new Zombie();
        //cout << "Name" << i + 1 << ": " << zom[i]->show_name() << endl;
    }
    int win = 0;
    int turn = 0;
    int current = 0; // Count current zombie
```

```
// Game loop
while (win != 99) {
    turn += 1;
    cout << "Round # " << turn << endl;
    cout << "Zombie # " << current << endl;

    // Reload or Shoot/Hurt Zombies
    if (player.show_bullet() > 0 && current < 10) {
        // Check if the player wants to reload
        cout << "You have " << player.show_bullet() << " bullets left" << endl;
        cout << "Do you want to reload? (1 for yes, 0 for no): ";
        int reloadChoice;
        cin >> reloadChoice;

        if (reloadChoice == 1) {
            reload(player);
            cout << "Player skipped the turn to reload." << endl;
        }
        if (reloadChoice == 0) {
            if (player.show_bullet() > 0) {
                // Shoot or hit Zombies
                player.shoot(*zom[current]);
            }
            else{
                    player.hit(*zom[current]);
                }

            }

            // Zombie bite 30%
            zom[current]->bite(player);

            // Display Zombie and player information
            cout << "Player Health: " << player.show_health() << " | ";
            cout << zom[current]->show_name() << " Health: " << zom[current]->show_health() << endl;
```

```
            // Move to the next zombie if the current one is defeated
            if (zom[current]->show_health() <= 0) {
                current++;
            }

    }


    // Check if all Zombies are defeated
    if (current == 10) {
        cout << "Congratulations! All Zombies are defeated. You win!" << endl;
        win = 99;
    }

    // Check if the player is still alive
    if (player.show_health() <= 0) {
        cout << "Game over! The player has been defeated." << endl;
        win = 99;
    }
}
//delete zoms
for (int i = 0; i < 10; ++i) {
    delete zom[i];
}

return 0;
```

Explaination: In the main I include the header files. The I have the funtions that will be used in the main, The heal function is incomplete here I would set the zombie and human classes health to 100 randomly every 5 rounds. The reload function adds 5 bullets to the total, if it goes over max 7 then resets to 7. The main starts off creating human class and creating 10 zombies on the stack with a for loop. Then there is a while loop that controls the whole game, if any of the win conditions are met it prints who is the winner and ends the game. It starts counting the number of turns and current zombie in the loop so the current zomie can be increased. It uses an if condition to check if player has bullets and the zombies are less than 10. Option for reloading, or not reloading. Then zombie has a 30% chance to bite player. Calling a class method that is on the stack uses an -> arrow. Then displays zombie and player heath and checks if the zombie is defeated before moving onto the next zombie and round. Checks if the player or zombie won, if yes exits program and displays winner. Deletes zombies saved on stack.

Human.h:



```h
#pragma once
#include <iostream>
#include <string>
#include "Zombie.h"
using std::string;
class Human
{
    //attribute
    string name;
    int health;
    int bullet;
    friend class Zombie;
    friend void reload(Human& user);

public:
    //constructor (set defualt variable)
    Human();
    Human(int in_health, string in_name, int bullet);
    //methods
    string show_name();
    int show_health();
    int show_bullet();
    void shoot(Zombie& user);
    void hit(Zombie& user);

    //destructor
    ~Human();
};
```

Explaination: The Human header, the attributes are name, health and bullet. It is made friend with Zombie class so it can affect its health. It is made friends with reload function so it can change bullet amont. The constructors set defualt or use input. The methods display name and health and amount of bullets. Has ability to shoot and hit which will inflict damage to zombie.

Human.cpp

```cpp
#include "Human.h"

//constructor (set defualt variable)
Human::Human() {
    name = "Human";
    health = 100;
    bullet = 7;
}
Human::Human(int in_health, string in_name, int in_bullet) {
    name = in_name;
    health = in_health;
    bullet = in_bullet;
}
//methods
string Human::show_name() {
    return name;
}
int Human::show_health() {
    return health;
}
int Human::show_bullet()
{
    return bullet;
}
void Human::shoot(Zombie& user) {
    if (bullet > 0) {
        bullet -= 1;
        if (user.weak() == "head" || user.weak() == "heart") {
            user.health -= 50;
        }
        else {
            user.health -= 25;
        }
    }
}
void Human::hit(Zombie& user) {
    user.health -= 10;
    if (health < 0) {
        user.health = 0;
    }
}

//destructor
Human::~Human() {

}
```

Explaination: Has a defualt for the attributes, or input choice for the attrivutes to be set. Has methods to return its name health and bullets (no input) output of type to return. Method shoot uses user weakness function it will subtract a bullet when called and if it hits the weak point subtract 50 health or if it doesn't hit the current weakpoint it subtract 25 health for zombie by using Zombie& user since it is friends it has access to its attributes. Same as hit method where it subraccts the zombies health by 10.

Zombie.h:

```h
#pragma once
#include <iostream>
#include "Human.h"
#include <string>
using std::string;
class Zombie
{
    //attribute
    string name;
    int health;
    string weak_point;
    friend class Human;
public:
    //constructor
    string weak();
    Zombie();
    Zombie(string in_name, int in_health, string in_weak_point);
    //methods
    int show_health();
    void bite(Human& user);
    string show_name();
    //destructor
    ~Zombie();
};
```

Explanation: The zombie header has attributes of name, health, and weak point, and is friends with class human to affect their attributes. Has methods to show health and name and bite user.

Zombie.cpp:

```cpp
#include "Zombie.h"
string Zombie::weak() {
    string weakness;
    if (rand() % 2 == 0) {
        weakness = "head";
    }
    else {
        weakness = "heart";
    }
    return weakness;
}
Zombie::Zombie() {
    name = "Zombie";
    health = 100;
    weak_point = weak();
}
Zombie::Zombie(string in_name, int in_health, string in_weak_point) {
    name = in_name;
    health = in_health;
    weak_point = in_weak_point;
}
//methods
void Zombie::bite(Human& user) {
    if (rand() % 10 < 3) { // 30% chance of getting bitten
        user.health -= 10;
    }
}
string Zombie::show_name() {
    return name;
}
int Zombie::show_health() {
    return health;
}

//destructor
Zombie::~Zombie() {

}
```

Attributes: Has weak function that uses a random number generator to determine weak point. This works but is incorrect because it creates a new weak point every turn instead of a predetermined weak point. This can be fixed by making the weakpoint in the main. It has constructors of default values, or different constructor of input values. It has bite methods which is 30% random chance of biting user for -10 health uses Human& user to access Human attributes.

Results:

```
Round # 1
Zombie # 0
You have 7 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 90 | Zombie Health: 50
Round # 2
Zombie # 0
You have 6 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 80 | Zombie Health: 0
Round # 3
Zombie # 1
You have 5 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 80 | Zombie Health: 75
Round # 4
Zombie # 1
You have 4 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 70 | Zombie Health: 25
Round # 5
```

```
Do you want to reload? (1 for yes, 0 for no): 1
Player skipped the turn to reload.
Player Health: 10 | Zombie Health: 50
Round # 25
Zombie # 7
You have 7 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 10 | Zombie Health: 0
Round # 26
Zombie # 8
You have 6 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 10 | Zombie Health: 50
Round # 27
Zombie # 8
You have 5 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 10 | Zombie Health: 25
Round # 28
Zombie # 8
You have 4 bullets left
Do you want to reload? (1 for yes, 0 for no): 0
Player Health: 0 | Zombie Health: -25
Game over! The player has been defeated.
```

Self-assessment: This lab really developed my skills with classes. I learned that you could add a function as a friend from the main.cpp to affect a class as used in the reload function. I had to work around some errors, I should've planned my logic as I had an issue with the weak point explained earlier. I should've done this different by predetermining the weak point. I also could've made a better user interface. However, this game was a learning experience and in the future labs I will be able to perfect it.