

/***** *

- Author: Samuel Campbell
- Email: Sccampbell1019@my.msutexas.edu
- Label: P03A
- Title: Rock Paper Scissors Lizard Spock
- Course: CMPS 2143
- Semester: Fall 2021
-
- Description:

- A rock paper scissors game that uses overloaded operators to compare players hands

- using a map and overloaded operators to determine a winner.

-
- Usage:

- srand(time(0));

- for(int i=0; i<26; i+=2)

- {

- Player p1;

- Player p2;

- (p1>p2);

- cout<<endl<<endl;

- }

-
- Files: rockpaper.cpp *****/

```
#include #include // needed for bind #include #include #include #include #include #include
```

```
using namespace std;
```

```
#define ROCK u8"\U0000270A" #define PAPER u8"\U0000270B" #define SCISSORS u8"\U0001F44C" #define LIZARD u8"\U0001F918" #define SPOCK u8"\U0001F596"
```

```
#define ROCK2 u8"\U0001F5FB" #define PAPER2 u8"\U0001F4C3" #define SCISSORS2 u8"\U0001F52A" #define LIZARD2 u8"\U0001F438" #define SPOCK2 u8"\U0001F596"
```

```
/**
 * Public : map< string, string > Weapons & map< string, string > Names
 *
 * Description:
 * enables us to use strings "rock", "paper", and such to call the emojis.
 *
 * Params:
 * <string, string>
 *
 * Returns:
 * N/A
 */
```

```
map< string, string > Weapons = { {"rock", ROCK2}, {"paper", PAPER2}, {"scissors", SCISSORS2}, {"lizard", LIZARD2}, {"spock", SPOCK2} };
```

```
map< string, string > Names = { {ROCK2, "rock"}, {PAPER2, "paper"}, {SCISSORS2, "scissors"}, {LIZARD2, "lizard"}, {SPOCK2, "spock"} };
```

```
/**
 * Public : map <string , vector<string>> rules
 *
 * Description:
 * This rules map provides the rules for the game.
 * The vector<string> contains the hands that are beaten
 * by the left most string
 *
 *
 */
```

```

* Params:
*     <string, vector<string>>

*
* Returns:
*     N/A
*/

```

```

map <string , vector> rules = { {"rock", {"lizard","scissors"} }, {"paper", {"rock","spock"} }, {"scissors", {"paper","lizard"}}, {"lizard", {"spock","paper"}}, {"spock", {"rock","scissors"}} };

```

```

/**
* Public : string RandWeapon()
*
* Description:
*   This function iterates the Weapon map and travels a
*   random amount and grabs and returns an emoji.
*
* Params:
*   none
*
* Returns:
*   string random_weapon
*/

```

```

string RandWeapon() { auto it = Weapons.begin(); // iterator to front of map

```

```

    std::advance(it, rand() % Weapons.size()); // advance some random amnt
                                              //   of steps
    string random_weapon = it->first; // grab emoji from map
    return random_weapon; // return rand emoji

```

```

}

```

```

/**
* Public : bool beats()
*
* Description:
*   This function iterates the rules map and travels till
*   it finds the hand1 and then looks at the vector<string, string>
*   to see if it finds hand2 in that vector. If so it returns True
*   if it finds it. This means that if true then Hand1 beats Hand2.
*
* Params:
*   string    hand1

```

```

*      string    hand2
*
* Returns:
*      bool
*/

```

```

bool beats(string hand1, string hand2){ auto it = find (rules[hand1].begin(), rules[hand1].end(), hand2); if (it !=
rules[hand1].end()){ return 1; } return 0; }

```

```

/**

```

- Class Weapon
-
- Description:

- contains the constructor for weapon so that 'w = string name'

- contains the overloaded operators that allow us to compare the

- two hands.

- Private: string name

- friend class Player

- Public Methods:

- - Weapon()

- - Weapon(string w)

- - friend ostream& operator<<(ostream &os,const Weapon &w)

- - bool operator>(const Weapon &rhs)

- - bool operator==(const Weapon &rhs)

- Usage:

-
-
-

*/ class Weapon{ string name; friend class Player; public:

```
/**
 * Public : Weapon()
 *
 * Description:
 *   Constructor for Weapon()
 *
 * Params:
 *   none
 *
 * Returns:
 *   N/A
 */
```

Weapon(){ name = RandWeapon(); }

```
/**
 * Public : Weapon(string w)
 *
 * Description:
 *   Constructor for Weapon() and sets 'name = w'
 *
 * Params:
 *   string w
 *
 * Returns:
 *   N/A
 */
```

Weapon(string w){ name = w; }

/**

- Public : operator<<(ostream &os,const Weapon &w)
-
- Description:
- Overloading << operator so when a weapon prints it prints the name

- and that name will appear as an emoji
-
- Params:
 - ostream &os,
 - const Weapon &w
-
- Returns:

- `friend os << Weapon[w.name]`

```
*/ friend ostream& operator<<(ostream &os,const Weapon &w){ return os << Weapons[w.name]; }
```

```
/**
```

- Public : operator>(const Weapon &rhs)
-
- Description:
 - called to compare two hands. Returns true if this->name
 - beats rhs.name
-
- Params:
 - const Weapon &rhs
 - const Weapon &w
-
- Returns:

- `bools`

```
*/ bool operator>(const Weapon &rhs ){ if(beats(this->name, rhs.name)){ return 1; } return 0; }
```

```
/**
```

- Public : operator==(const Weapon &rhs)
-
- Description:
 - called to compare two hands. Returns true if
 - both this->name and rhs.name are the same
-
- Params:
 - const Weapon &rhs
 - const Weapon &w
-
- Returns:

- bools

```
*/ bool operator==(const Weapon &rhs ){ if(this->name == rhs.name){ return 1; } return 0; };
```

```
/**
```

- Class Player:Weapon
-
- Description:

- contains the constructor for weapon so that 'w = string name'

- contains the overloaded operators that allow us to compare the

- two hands.

- Private: Weapon primary

- Weapon secondary

- Public Methods:

- - Player()

- - Player(string w1, string w2)

- - friend ostream& operator<<(ostream &os,const Weapon &w)

- - bool operator>(const Weapon &rhs)

- - bool operator==(const Weapon &rhs)

-
- Usage:

-

- `Player p1;`

- `Player p2;`

- `(p1>p2);`

```
*/ class Player:Weapon{ Weapon primary; Weapon secondary;
```

```
public:
```

```
/**
```

- Public : Player()
-
- Description:
- constructor for player
- assigns random weapons for Primary and Secondary
- and if they are the same then it will change the
- secondary weapon.
- Params:
- none
-
- Returns:

- `N/A`

```
*/ Player(){ // random primary and secondary primary = RandWeapon(); secondary = RandWeapon(); while
(primary.name == secondary.name){ secondary = RandWeapon(); } }
```

```
/**
```

- Public : Player(string w1,string w2)
-
- Description:
- constructor for player
- assigns weapons for Primary and Secondary
- and if they are the same then it will change the
- secondary weapon.

- Params:
- -string w1
- -string w2
-
- Returns:

- N/A

```
*/ Player(string w1,string w2){ primary = w1; secondary = w2; // both weapons assigned while (primary.name
== secondary.name){ // insures variety of weapons secondary = RandWeapon(); // for each player }
}
```

/**

- Public : operator>(const Player& other)
-
- Description:
- Used to determine the winner. While it is a bool, instead of
- printing the bool result,it prints a message depending on who wins,
- instead of printing the bool result.
-

- Params:
- ◦ const Player&

- ◦ other

- Returns:

- bool

```
*/ bool operator>(const Player& other){ // check if they equal first while(this->primary == other.primary){
this->primary = RandWeapon(); } while(this->secondary == other.secondary){ this->secondary =
RandWeapon(); } if(this->primary > other.primary){ cout << "primary:"<< this->primary << " "; cout <<
"secondary:"<< this->secondary << endl; cout << "primary:"<< other.primary << " "; cout << "secondary:"
<< other.secondary << endl; cout << "Player 1's Primary:"<<primary<<" Beats Player 2's Primary:"<<
other.primary <<endl; return true; }else if(this->secondary > other.secondary){ cout << "primary:"<< this-
>primary << " "; cout << "secondary: "<< this->secondary << endl; cout << "primary:"<< other.primary << "
"; cout << "secondary:"<< other.secondary << endl; cout << "player 1's Secondary:"<<secondary<<" Beats
Player 2's Secondary:"<< other.secondary <<endl; return true; } else cout << "primary:"<< this->primary << "
"; cout << "secondary:"<< this->secondary << endl; cout << "primary:"<< other.primary << " "; cout <<
"secondary:"<< other.secondary << endl; cout << "Player 2's Primary:"<<other.primary<<" Beats Player 1's
Primary:"<< this->primary <<endl; return false; } };
```

```
int main() { srand(time(0)); for(int i=0; i<26; i+=2) { Player p1; Player p2; (p1>p2); cout<<endl<<endl; } return 0; }
```