# Assignment Description

In this program, you will create an object and display it. Create a class nerfGun that includes data members model (string), range(int), capacity (int), and number of darts (int). Your class should have a constructor with three parameters (assume the toy is fully loaded when created). Assume that model and range is correct, but check to make sure capacity is <= 144. Provide a member function getModel(), getCapacity(), getDartCount(), fire() and reload(quantity). The method fire() reduces the number of darts (throw an underflow exception if the gun is fired after the inventory is exhausted. The method reload(quantity) should increase the number of darts by the quantity provided but must ensure the number of darts do not exceed the toy’s capacity (throw an overflow exception) .

1. Create overloaded operators:
   1. bool operator == that overloads the == operator to check if two nerf guns have the same amount of ammunition.
   2. bool operator < that overloads the < operator and checks to see if one gun has less ammunition than the other.
   3. Overload the -– operators (both prefix and postfix) to fire the gun (and decrease the ammunition). Throw an appropriate exception if an attempt is made to fire a gun without remaining ammunition.
   4. Overload the += operator to allow the gun to be reloaded (by adding the specified number of rounds). Throw an appropriate exception if someone attempts to add more ammunition than the gun can handle.
   5. Overload the stream insertion operator << to allow the guns information to be printed:

cout << myGun;

Create a program main() that will do the following:

* Create a vector to contain nerfGuns.
* Create 4 to 7 different guns.
* Display each gun’s information.
* Compare the guns using the overloaded operator and display which one has the most ammunition.
* Use the overloaded operators to fire and reload the guns randomly, displaying their status/information.
* Handle any exceptions for overloading or exhausting the guns ammunition.

# 1 Readme Documentation

This program will create 4 nerfGun objects. It will then randomly fire and reload these guns, displaying their new states and appropriately handling over and underflow errors.

# 2 Flowchart Screen Shots

# 3 UML and Use Case Diagrams

# 4 Source Code of All files (.h, .cpp)

#include *<iostream>*

#include *<iomanip>*

#include *<string>*

#include *<cctype>*

#include *<stdexcept>*

#include *<vector>*

#include *<fstream>*

#include *<random>*

#include *<ctime>*

#include *<array>*

**using** **namespace** **std**;

**class** **nerfGun** {

**private**:

string model;

int range;

int capacity;

int dartCount;

**public**:

*// Constructor*

nerfGun(string inputModel, int inputRange, int inputCapacity){

setModel(inputModel);

setRange(inputRange);

setCapacity(inputCapacity);

*// Fills clip with darts*

dartCount = capacity;

}

*// Default constructor*

nerfGun(){}

*// Mutator Methods*

void setModel(string inputModel){model = inputModel;}

void setRange(int inputRange){

**if**(inputRange < 1) **throw**(invalid\_argument("Range cannot be less than 1"));

range = inputRange;

}

void setCapacity(int inputCapacity){

**if**(inputCapacity > 144) **throw**(invalid\_argument("Capacity cannot be greater than 144"));

**if**(inputCapacity < 1) **throw**(invalid\_argument("Capacity cannot be less than 1"));

capacity = inputCapacity;

}

void fire(){

**if**(dartCount == 0) **throw**(underflow\_error("Cannot fire with no darts in the clip"));

*// Shoot one dart*

dartCount -= 1;

}

void reload(int reloadAmount){

int spaceLeft = capacity - dartCount;

*// Throws exception if not enough space in clip for reload amount*

**if**(reloadAmount > spaceLeft) **throw**(overflow\_error("Not enough space in " + getModel() + " to reload " + to\_string(reloadAmount) + " darts."));

**if**(reloadAmount < 0) **throw**(invalid\_argument("Reload amount cannot be less than 0"));

*// Reloads clip*

dartCount += reloadAmount;

}

string toString() **const** {

string gunString;

string line(75, '-');

string newline;

int width = 74;

gunString += '\n' + line;

newline = "**\n**| Model: " + getModel();

newline += string(width - newline.length(), ' ') + " |";

gunString += newline;

gunString += '\n' + line;

newline = "**\n**| Range: " + to\_string(range) + " Feet";

newline += string(width - newline.length(), ' ') + " |";

gunString += newline;

gunString += '\n' + line;

newline = "**\n**| Capacity: " + to\_string(capacity) + " Darts";

newline += string(width - newline.length(), ' ') + " |";

gunString += newline;

gunString += '\n' + line;

newline = "**\n**| Dart Count: " + to\_string(dartCount) + " Darts";

newline += string(width - newline.length(), ' ') + " |";

gunString += newline;

gunString += '\n' + line + '\n';

**return** gunString;

}

*// Accessor Methods*

*// Const because they don't need to change anything*

string getModel() **const** {**return** model;}

int getCapacity() **const** {**return** capacity;}

int getDartCount() **const** {**return** dartCount;}

int getRange() **const** {**return** range;}

*// Operator Overloading*

*// Returns true if both guns have same dart count*

bool **operator**==(**const** nerfGun& otherGun){

**if**(dartCount == otherGun.getDartCount()){

**return** true;

}

**else**{**return** false;}

}

*// Returns true if first gun has less darts than second gun*

bool **operator**<(**const** nerfGun& otherGun){

**if**(dartCount < otherGun.getDartCount()){

**return** true;

}

**else**{**return** false;}

}

nerfGun **operator**--(){

nerfGun temp = \***this**;

fire();

**return** temp;

}

*// Fires gun and throws exception if dartCount == 0*

nerfGun **operator**--(int){

nerfGun temp = \***this**;

fire();

**return** temp;

}

*// Reloads gun and throws exception if reload amount is too much or negative*

nerfGun **operator**+=(**const** int& reloadAmount){

nerfGun temp = \***this**;

reload(reloadAmount);

**return** temp;

}

*// Declares stream extraction operator friend function*

**friend** ostream& **operator**<<(ostream& output, **const** nerfGun& gun);

};

*// Declared outside nerfGun methods so that cout << gun works*

ostream& **operator**<<(ostream& output, **const** nerfGun& gun){

output << endl << gun.toString();

**return** output;

}

void displayGuns(**const** vector<nerfGun> &guns);

nerfGun mostAmmunition(**const** vector<nerfGun> &guns);

int menu();

int main(){

vector<nerfGun> guns;

*// Creating guns and adding to vector*

nerfGun gun1("gun1", 15, 6);

guns.push\_back(gun1);

nerfGun gun2("gun2", 10, 20);

guns.push\_back(gun2);

nerfGun gun3("gun3", 13, 5);

guns.push\_back(gun3);

nerfGun gun4("gun4", 10, 15);

guns.push\_back(gun4);

cout << "The following is the generated Nerf guns: ";

*// Display guns*

displayGuns(guns);

*// Finds the gun with the most ammo in its clip*

nerfGun mostAmmo = mostAmmunition(guns);

cout << endl << "The gun with the most ammo is:" << mostAmmo << endl;

srand(time(0));

**for**(int i = 0; i < 10; i++){

**try**{

int gunIndex = rand() % guns.size();

int action = rand() % 2;

nerfGun \*randGun = &guns[gunIndex];

**switch**(action){

**case** 0:

*// Fire a random gun*

(\*randGun)--;

cout << "Fired gun: " << \*randGun;

**break**;

**case** 1:

*// Reload a random gun*

(\*randGun) += 1;

cout << "Reloaded gun: " << \*randGun;

**break**;

}

}**catch**(**const** exception& e){

cout << endl << e.what() << endl;

}

}

**return** 0;

}

*// Displays all nerf guns using pointers*

void displayGuns(**const** vector<nerfGun> &guns){

*// Iterates through guns vector with pointer*

**for**(**auto** ptr = guns.begin(); ptr != guns.end(); ptr++){

*// Outputs each gun*

nerfGun gun = \*ptr;

cout << endl << gun;

}

}

nerfGun mostAmmunition(**const** vector<nerfGun> &guns){

int max = 0;

nerfGun mostAmmo;

mostAmmo = guns[0];

**for**(**auto** ptr = guns.begin(); ptr != guns.end(); ptr++){

nerfGun gun = \*ptr;

*// Compares dart count of guns to find max*

**if**(mostAmmo < gun){

max = gun.getDartCount();

mostAmmo = gun;

}

}

**return** mostAmmo;

}

# 5 Three Use Case Screen Shots























