NTUST OOP Midterm Problem Design

Subject: Buff System Extended

Contributor: Yen-Chen Chiu , Kevin W.

Main testing concept:

Basics	Functions
■ C++ BASICS	□ SEPARATE COMPILATION AND NAMESPACES
■ FLOW OF CONTROL	■ STREAMS AND FILE I/O
■ FUNCTION BASICS	□ RECURSION
■ PARAMETERS AND OVERLOADING	□ INHERITANCE
■ ARRAYS	□ POLYMORPHISM AND VIRTUAL FUNCTIONS
■ STRUCTURES AND CLASSES	□ TEMPLATES
■ CONSTRUCTORS AND OTHER TOOLS	□ LINKED DATA STRUCTURES
□ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES	□ EXCEPTION HANDLING
■ STRINGS	□ STANDARD TEMPLATE LIBRARY
□ POINTERS AND DYNAMIC ARRAYS	□ PATTERNS AND UML

Description:

Most classical RPGs (Role-Play Games) have BUFF system. BUFF means "Beneficial Effect", which is an effect placed on a character that enhances their statistics or characteristics. There're negative effects called DEBUFFs, which reduce statistics or characteristics.

In this test, a character and a BUFF should at least have these methods:

// This enum is already declared in <attribute.h>

```
Enum Attribute
```

- `Power`
- `Defense`
- `Speed`
- `Empty (Optional)`

// Provided in <Player.h>

```
Class Player
□ float Power, Defense, Speed
□ constructor (float power, float defense, float speed)
□ void parse()
```

- A character can have multiple BUFFs (less than 50), and BUFFs can be varied. BUFFs may affect different attributes. E.g., one can control the power, another controls defense, depends on the attribute property.
- Note that, one BUFF only affects one attribute of a character, in other words, it affects either power, defense, or speed. Applying one BUFF means multiply the attribute by multiplier first, then plus by addend. E.g., **(power * multiplier) + addend**.
- Every BUFF has their priority, the higher priority number is, the earlier it gets applied. No matter what time the BUFF is added on the character. For example, assume there are BUFF A and B, where A.priority is 5, B.priority is 3, they all affects the power attribute. So the character's power would be:

  (((power * A.multiplier) + A.addend) * B.multiplier) + B.addend

  No matter A or B is added on the player first.
- No two BUFFs with the same priority that modifies the same attribute would be applied.

□ `void addBuff(Buff buff)`

- If a new effect with the same name applied, the remaining time will be set to the longer remaining time.
- No effect with the same name but different Attribute, Priority, Addend, Multiplier would be applied.
- It takes 1s for a new affect to be added, in other words, all applied BUFFs remaining time would be subtracted by 1.
- A BUFF is considered as a DEBUFF when:

✧ Addend and multiplier both provides a negative modification to the original value
✧ Addend is 0, multiplier provides a negative modification.
✧ Multiplier is 1, addend provides a negative modification.

```
// Not a debuff, multiplier > 1
A.multiplier = 1.1, A.addend = -999
// Not a debuff, addend > 0
A.multiplier = 0.01, A.addend = 99999
// Not a debuff, no negative modification made
A.multiplier = 1, A.addend = 0

// Debuff
A.multiplier = 0.9, A.addend = -100 // rule 1
A.multiplier = 0.9, A.addend = 0    // rule 2
A.multiplier = 1, A.addend = -9999  // rule 3
```

□ `void removeBuff(string name)`

- Removes <name> BUFF instantly.

□ `void cleanse()`

- A special instant effect, which removes all DEBUFFs.
- Does not take time to be applied.

□ `void tick(int time)`

- Skip <time>, remove <time> duration from all remaining time.

// Provided in <Buff.h>

`Class Buff`

□ `constructor (Attribute type, string name, int priority, float addend, float multiplier, int duration)`

- Construct a buff via its type, name, priority, addend, multiplier, and duration.

Input:

First line of the input is the base character's attributes, with order of power, defense, and speed.

□ These numbers are between 1 to 100000.

Then for the following multiple lines, each line is a command either of these 5 types:

1. Add Buff, in format of: "**add <effect type> <name> <priority> <addend> <multiplier> <duration>**"
2. Remove Buff, in format of: "**remove <name>**
3. Cleanse, in format of : "**cleanse**"
4. Tick time, in format of: "**tick <time>**"
5. Parse the character's attribute, in format of: "**parse**" in output description.

□ The name of a BUFF is only a combination of letters, includes upper and lower case, and it is case sensitive, will not contains spaces or other symbols.

□ A Player is allowed to have negative power, defense, and speed.

□ There won't be two BUFFs inputted with same type AND same priority.

□ **No commands with error / missing parameters will be input in the testing data.**

□ **main.cpp will be REPLACED, do not edit!**

Output:

□ When the command is "**add**", and the BUFF duration is extended (See rule 1 for addBuff), print "**Add BUFF <Buff name> extended!(\n)**", otherwise, print "**Add BUFF <Buff name> success!(\n)**"

□ When the command is "**remove**", and the name cannot be found in the buff list, print "**Remove BUFF <Buff name> failed!(\n)**", otherwise, print "**Remove BUFF <Buff name> success!(\n)**"

□ When the command is "**cleanse**", print "**Cleanse: <BuffName1>, <BuffName2>….(\n)**", if no BUFF is removed, print "**Cleanse nothing(\n)**" where BuffName is BUFFs removed in priority order, if they have the same priorities, order them by their type, power first, then defense, and speed.

□ When the command is "**tick**", nothing is printed.

□ When the command is "**parse**", prints all the character's attributes:

1. power
2. defense
3. speed
4. list of buffs' names separated by commas and space. Order by priority from high to low,

(a) if they have the same priorities, order them by their type, power first, then defense, and speed.

(b) If it's empty, prints "**No Buff(\n)**"

Format:
Power: 45(\n)
Defense: 15(\n)
Speed: 4(\n)
Buff List: <BuffName1>, <BuffName2>, ….(\n)

See Sample output for exact result.

Sample Input	Sample Output
600 10 100 parse add power Rage 10 100 1 10 add speed Swiftness 10 150 1 10 parse add speed Lightspeed 11 0 2 20 add power Strength 11 0 2 20 parse add power heroic 12 10000 1 1 add speed lightspeed 12 10000 1 1 parse	Power: 600 Defense: 10 Speed: 100 No Buff Add BUFF Rage success! Add BUFF Swiftness success! Power: 700 Defense: 10 Speed: 250 Buff List: Rage, Swiftness Add BUFF Lightspeed success! Add BUFF Strength success! Power: 1300 Defense: 10 Speed: 350 Buff List: Strength, Lightspeed, Rage, Swiftness Add BUFF heroic success! Add BUFF lightspeed success! Power: 1300 Defense: 10 Speed: 20350 Buff List: lightspeed, Strength, Lightspeed, Rage, Swiftness

100 100 10 add defense Resistance 10 10 1.5 10 add power Rage 10 20 1.2 10 add speed Swiftness 10 5 1.2 10 parse add speed Lightspeed 11 10 2 20 add power Strength 11 30 2 20 add defense Hardened 11 100 2 20 parse add speed StickyGround 8 -1 0.9 10 add defense Fragile 9 -10 0.8 10 add power Weakness 9 -50 1 10 parse cleanse parse remove Strength remove Saturation parse tick 10 parse add defense Hardened 11 100 2 20 tick 10 parse	Add BUFF Resistance success! Add BUFF Rage success! Add BUFF Swiftness success! Power: 140 Defense: 160 Speed: 17 Buff List: Rage, Resistance, Swiftness Add BUFF Lightspeed success! Add BUFF Strength success! Add BUFF Hardened success! Power: 296 Defense: 460 Speed: 41 Buff List: Strength, Hardened, Lightspeed, Rage, Resistance, Swiftness Add BUFF StickyGround success! Add BUFF Fragile success! Add BUFF Weakness success! Power: 246 Defense: 358 Speed: 35.9 Buff List: Strength, Hardened, Lightspeed, Rage, Resistance, Swiftness, Weakness, Fragile, StickyGround Cleanse: Weakness, Fragile, StickyGround Power: 296 Defense: 460 Speed: 41 Buff List: Strength, Hardened, Lightspeed, Rage, Resistance, Swiftness Remove BUFF Strength success! Remove BUFF Saturation failed! Power: 140 Defense: 460 Speed: 41 Buff List: Hardened, Lightspeed, Rage, Resistance, Swiftness Power: 100 Defense: 300 Speed: 30 Buff List: Hardened, Lightspeed Add BUFF Hardened extended! Power: 100 Defense: 300 Speed: 10 Buff List: Hardened

□ **Easy, Only basic programming syntax and structure are required.**
□ **Medium, Multiple programming grammars, and structures are required.**
■ **Hard, Need to use multiple program structures or complex data types.**

Expected solving time:

50 min.

Other Notes: