**10902 CPP Midterm Exam**

|  |
| --- |
| **Contributor︰Yen-Chen Chiu** |
| **Subject：BUFF System** |
| **Main testing concept：**   |  |  | | --- | --- | | **Basics** | **Functions** | | ■ C++ BASICS  ■ FLOW OF CONTROL  ■ FUNCTION BASICS  ■ PARAMETERS AND OVERLOADING  ■ ARRAYS  ■ STRUCTURES AND CLASSES  ■ CONSTRUCTORS AND OTHER TOOLS  □ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES  □ STRINGS  ■ POINTERS AND DYNAMIC ARRAYS | □ SEPARATE COMPILATION AND NAMESPACES  □ STREAMS AND FILE I/O  □ RECURSION  ■ INHERITANCE  ■ POLYMORPHISM AND VIRTUAL FUNCTIONS  □ TEMPLATES  □ LINKED DATA STRUCTURES  □ EXCEPTION HANDLING  □ STANDARD TEMPLATE LIBRARY  □ 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. But in this test, we take these all as same as BUFF, no matter the effect is positive or negative.  In this test, a character and a BUFF should at least have these properties:  (**main.cpp will be replaced for testing your implementation.**)   |  |  | | --- | --- | | Class: Character | Class: Buff | | * float: power * float: defense * float: speed * array of Buffs: buffList * constructor (float power, float defense, float speed) * function: Parse() * function: AddBuff(Buff buff) * function RemoveBuff(string name) | * string: type * string: name * integer: priority * float: addend * float: multiplier * constructor (string type, string name, int priority, float addend, float multiplier) |  * 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. * 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, and 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.   * Note that, after each calculation of one BUFF, the attribute number would round up to nearest integer. (e.g., 12.7 to 13, 12.4 to 12) * Also, a character will NOT be able to have any two BUFFs with same name at same time.  If a new BUFF with existing name is trying to add, the new one would be discarded.   **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 10000.   Then for the following multiple lines, each line is a command either of these 3 types:   1. Add Buff, in format of: “add {affect type} {name} {priority} {addend} {multiplier}” 2. Remove Buff, in format of: “remove {name}” 3. 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. * All the buffs’ numbers are between -100 to +100. * There won’t be two BUFFs inputted with same type AND same priority.   **Output：**   * When the command is “add”, and if the buff has same name in the buff list,  prints “Add BUFF Failed!” * When the command is “remove”, and if the name cannot be found in the buff list,  prints “Remove BUFF Failed!” * 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,    1. if there have same priorities, ordered by the Buffs’ types,  power first, defense at second, speed last.    2. If it’s empty, prints “No Buff”   **Sample Input / Output :**   |  |  | | --- | --- | | **Sample Input** | **Sample Output** | | 100 50 10  parse  add power A 5 0 2  add power B 7 10 1  parse  remove A  parse | Power: 100  Defense: 50  Speed: 10  Buff List: No Buff  Power: 220  Defense: 50  Speed: 10  Buff List: B, A  Power: 110  Defense: 50  Speed: 10  Buff List: B | | 200 70 18  add power BattleCry 9 0 1.5  add defense HardShield 4 50 2  add speed Winds 1 0 1.2  parse  add power Rage 5 20 2  remove HardShield  add speed Winds 3 0 1.3  add defense IronWill 5 30 1  remove HardShield  remove BattleCry  parse | Power: 300  Defense: 190  Speed: 22  Buff List: BattleCry, HardShield, Winds  Add BUFF Failed!  Remove BUFF Failed!  Power: 420  Defense: 100  Speed: 22  Buff List: Rage, IronWill, Winds | | 90 40 30  add defense Scared 9 -10 1  add power Tired 8 0 0.5  add speed Depressed 7 -5 0.3  add defense Hungry 6 0 0.5  parse  add power Relaxed 5 50 1.2  add defense Burger 4 30 1  parse  remove Hungry  remove Tired  add defense CatMemes 3 20 1  parse  remove Scared  remove Depressed  add speed LittleDepressed 10 -3 1  parse | Power: 45  Defense: 15  Speed: 4  Buff List: Scared, Tired, Depressed, Hungry  Power: 104  Defense: 45  Speed: 4  Buff List: Scared, Tired, Depressed, Hungry, Relaxed, Burger  Power: 158  Defense: 80  Speed: 4  Buff List: Scared, Depressed, Relaxed, Burger, CatMemes  Power: 158  Defense: 90  Speed: 27  Buff List: LittleDepressed, Relaxed, Burger, CatMemes | |
| **□** **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 minutes |
| **Other notes:**  **You can try a cooler design by making a BUFF as parent class, and for each different type of BUFF, derives it as the child class. It’s not the only way to implement this test, but it’s good for a game development.** |