Схващаш основната идея на задачите, но пропускаш да направиш някои важни неща и не се замисляш за особените ситуации, които могат да възникнат при употреба на класовете. На места усложняваш нещата малко повече.

1. Научи референциите, Сашо. Пак подаваш аргументите на функциите с копиране. Може би ще ме разбереш най-добре, ако ти го напиша в прав текст. Подавай аргумента като:

* const референция – в общия случай, когато аргументът няма да се променя
* обикновена референция – когато аргументът ще се променя
* указател – когато се налага (например ако подаваме масив)
* копие – само ако аргументът е от примитивен тип (int, char, unsigned, bool…) и те мързи :)

1. За нито един клас не си написал конструктор за копиране, нито оператор =. Един клас по принцип е незавършен без тях, но ако използваш динамична памет, както в BrowserHistory, те са абсолютно задължителни (както и деструктора), ако искаш класът да работи правилно.

Изобщо, решиш ли да пишеш конструктори за някой клас, е най-добре:

* ако той не използва динамична памет, напиши (поне) конструктор по подразбиране, конструктор за копиране и оператор =. От деструктор, в този случай, нямаш нужда
* ако той използва динамична памет, напиши и деструктор

1. Конструкторите за Word, които все пак си написал, не са правилни. Твоят Word() създава 21 неинициализирани символа (които не са ‘\0’ по подразбиране) и ако се опиташ да принтираш думата, създадена с този конструктор, най-вероятно ще видиш странни символи в конзолата. По-смислено е да създава празен низ, което ще стане, ако сложиш една ‘\0’ на нулевата позиция в масива.

Конструкторът Word (char\* \_word, int n) пък позволява да се напишат неща като:

Word w("Hello", 42);

Word w2("Haha", -5);

Не трябва да разчиташ, че програмите, които пишеш, ще бъдат използвани правилно. Потребителят трябва да има максимално малка „власт“ и безсмилици като горните два реда не трябва да са възможни. Вярно, това е задача за упражнение, но е хубаво да свикваш да мислиш по този начин. В конкретния случай проблемът е, че конструкторът приема два параметъра, за да присвои стойностите им на член-данни, които не са независими една от друга. По-безопасен конструктор тук би бил, например:

Word(char\* \_word) {

strcpy(word, \_word);

size = strlen(word);

}

Винаги, когато пишеш метод (и изобщо функция), трябва да се замислиш дали си предвидил възможностите за некоректно подаване на аргументи. В идеалният вариант класът трябва да е така написан, че обектът винаги да е в такова състояние, че да можеш да извикаш всеки метод, без това да предизвика грешка. Конструторите трябва да инициалицират всички член-данни. Зависими една от друга данни трябва да се променят синхронно.

1. int може да е отрицателно число, което не е ок, ако става дума, например, за индекс или размер на масив. (Редът websites = new HistoryEntry[\_n]; става напълно безсмислен, ако \_n e –12). Един вариант е да assert-ваш, както правиш на някои места, но обикновено е по-лесно и практично просто да използваш unsigned (или size\_t, макар че според мен unsigned върши чудесна работа). А и...
2. Assert-ването е добре да се използва само в краен случай и най-вече когато поведението на програмата е недефинирано (undefined behavior). В условието за Word е казано ако думата има 20 символа, операторите + и += да нямат ефект. Ти ги правиш да имат ефект и този ефект е: гръмнал компилатор. Трябваше да напишеш просто if(size >= 20) return; или нещо такова.

А операторът [ ] може да връща ‘\0’ при некоректен индекс – често е удобно в подобни ситуации да се връща празна стойност или стойност за грешка. Дали наистина е по-добре така, много зависи от общата картина – в някои проекти този вариант е по-подходящ, а в други е по-удачно да напишеш assert. Това го споменавам само като идея, в кратка задача за упражнение като тази няма правилен и грешен вариант.

Все пак, когато се изкушиш да напишеш assert, помисли дали няма друг начин, и ако не ти хрумне друга идея, тогава го напиши.

1. Премахването на елемент от масив може да се реализира доста лесно и без изтриване – стига да се сетиш, че може да не приемеш условието съвсем буквално. Методът removeResent спокойно можеше да е само един ред: curSize--;

Защо щеше да работи? Ами, защото навсякъде, където работиш с масива, ти гледаш само entry-тата с индекси 0, 1, ..., curSize – 1. И въпреки че информацията за url-то ще си стои в паметта, ти никога няма да я използваш и ще е все едно последното entry вече го няма.

Изкушаващо е да се мисли за обекта като за множество от член-данни, но това може да е подвеждащо. Състоянието на обекта всъщност се определя не от вътрешните данни, а от интерфейса (т.е. публичните методи). Не е толкова важно какво има в паметта, а какво ще се случи, ако се извика всеки от методите. Член-данните са само помощно средство за реализирането на интерфейса. Задачата е решена, ако направиш така, че класът да се държи по желания начин.

Ако много те притеснява оставянето на старата информация в паметта, може да направиш всички символи на изтритото url ‘\0’, а месеца да стане 0. Но заделянето на нова памет и копирането на цялата по-стара история са напълно неоправдани.

1. Степенуването е сложна операция, която е добре да се оптимизира по възможност.

Ти пресмяташ полинома по следния начин:

Работи, но така повтаряш някои сметки и извършваш повече операции от нужното. По-хитрата и икономична идея тук е:

Функцията pow също е добре да се избягва, когато се пресмятат цели числа, защото тя работи с double и връща отново double с всички произтичащи от това последствия – излишни каствания, загуба на точност...

1. По-оптимално е операторът + да вика +=, а не обратното, защото + създава нов обект, а това може да се спести при +=.
2. Не include-вай хедъри, когато не ти трябват, например assert.h и cstring в NumbersSummator.
3. При операторите < и == за Word не е достатъчно просто да сравниш всичко до края на първата дума – излиза, че “Java” == “JavaScript” и “Sasho” < “Sasho”.