本周我们将完成关于整数/字符串数组的查找、排序的三个小练习。目前为止关于字符串我们还没有讲得很多，对此你可以参考：

* c0 language reference中的第8章
* c0 library reference的第2.2节
* c0 Tutorial（<http://c0.typesafety.net/tutorial/Strings.html>）的关于字符串的那一页
* 或者在本文档的附录A中了解更多关于字符串的内容。

这次的任务代码发布在

<http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15122-s14/www/prog3.tgz>(译者注：点击这个网址只会下载本文档的英文原版，任务代码请查看老师发布的作业包)

代码中的文件README.txt复习了资料中的内容，解释了怎么上交作业。

对于你们上交作业到Autolab的时间没有限制。对于任务1-4我们会对你的代码进行输入输出反馈测试，测试案例对于找出漏洞很管用。对于1-4进行测试是技术选修科目。这些测试案例不会影响你的成绩。（Task5会影响你的成绩）（译者注：灰色部分都可以不用看，基本只适用于CMU学生）

# 第一大题 删除重复元素

在这个大题的编程练习中，您将获得一个已经排序好的字符串数组，您需要返回一个新的有序数组，其中包含的字符串没有重复的;也就是说，新数组将包含原数组中的每个字符串，但所有的字符串将是不同的。新数组的长度应足够大来容纳生成的字符串。

这个练习的代码请放在文件duplicates.C0里面。每个函数都要设置注释，包括前置条件(requires)，后置条件(ensures)和循环不变量(loop\_invariant)，也可以有一些必要的断言(assertation)。你可以把任何你需要的辅助功能函数放在文件中，但不能有main函数。

你的检测应该被放置在一个叫duplicates-test.c0的文件中，里面包含一个main函数。Autograder(自动评分器)会检验你的测试，但是这些测试不会被评分。

## Task 1 （1分）按照下面的声明(declaration)写一个函数

bool all\_distinct(string[] A, int n)

//@requires 0 <= n && n <= \length(A);

//@requires is\_sorted(A, 0, n);

其中n代表A的一个**子数组**的长度。在给出的数组所有元素（串）都不相同时，这个函数返回true，否则返回false

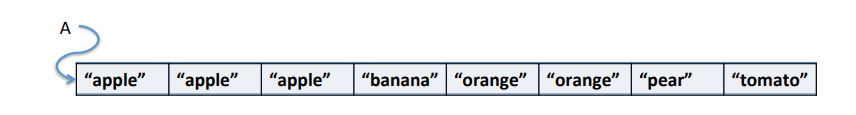
## Task2（1分）按照下面的声明(declaration)写一个函数

int count\_distinct(string[] A, int n)

//@requires 0 <= n&& n <= \length(A);

//@requires is\_sorted(A, 0, n);

其中n代表A的一个**子数组**的长度。这个函数能够返回这个数组中不同元素（串）的数目（如果同样的元素在这个数组中出现了不止一次，那么对于这个元素只记一次）。在这个数组中有五个水果：



因此count\_distinct(A,8)应该返回5。

程序的运行时间应该是线性的。认真思考：应该添加什么前提来减少这个函数的运行时间？（一个值得思考的问题：既然这个要求可以被满足，为什么我们不在all\_distinct里面使用它？）

## Task3（3分）按照下面的声明(declaration)写一个函数

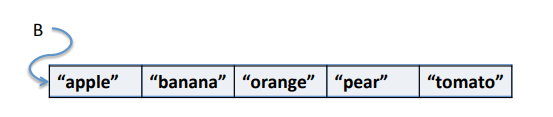
string[] remove\_duplicates(string[] A, int n)

//@requires 0 <=n && n <= \length(A);

//@requires is\_sorted(A, 0 ,n);

其中n代表A的一个**子数组**的长度。在传到这个函数之前，这个串需要先排序。这个函数应当返回一个仅包含原数组中每个不同元素的新数组，且新数组也得是有序的。

执行 string[] B = remove\_duplicates(A,8)应当返回这个数组

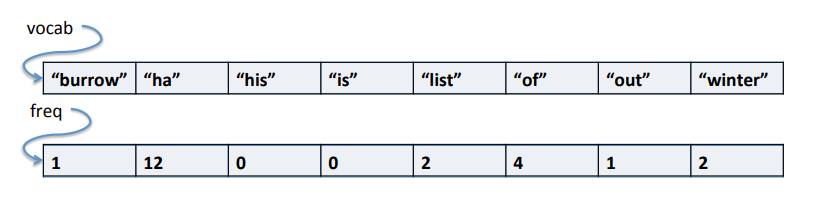


和函数count\_distinct一样，这个函数的运行的时间也应该是线性的。你的解答应该包括注释，注释中至少有3个强大的后置条件。

# 第二大题 DosLingos公司（计算相同单词）

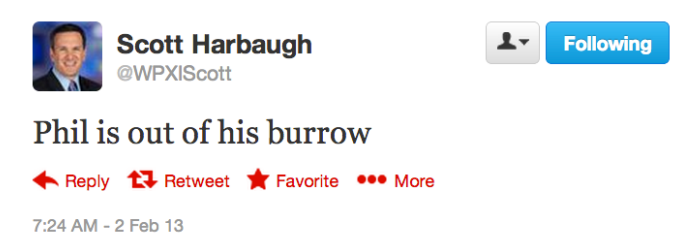
**背景故事**：你正在为一个处于创业阶段的自然语言处理公司DosLingos工作。你的公司已经成功地说服上千用户免费地把英文材料翻译成西班牙文(译者注：猜测应该是达成了某种合作关系，让用户帮忙进行翻译，将人的翻译结果和机器的翻译结果进行比对，以检测自然语言处理的正确性)。在近期的一次实验中，你让用户只翻译新闻稿，并成功地训练他们认识了英文报纸中的词汇。然而，你现在想让这批用户也翻译推特上的推文，但是你不确定只说西班牙语的用户能认出多少在英文推特中的方言。

**你的任务**：在这个练习中，你需要写一个函数来分析推特中有多少词汇出现在用户的词汇表中。用户词汇表是一个长度为v的有序数组vocab。我们还会维护另一个整数数组freq，freq下标为i的元素代表vocab[i]在推特文章中出现的次数。

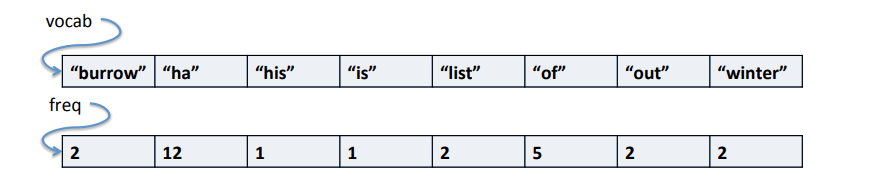


这是一个重要的模式，我们将在整个学期中的15-122不断地看到它：vocab中被排好序的词汇是*关键词*，存在freq中的次数是*值*。

我们写的函数count\_vocab根据推特数据中未排序的数据，不断更新这些关键词的值——即出现次数。举个例子，考虑一个推特文集仅包括下方这条推文（来自本地天气播报员 Scott Harbaugh）



则函数count\_vocab(vocab,freq,8,"texts/scottweet.txt",b)将返回1，（因为只有单词“Phil”不在样本词汇表vocab中），vocab中的内容保持不变，更新freq中的数据,如下：



**你的数据**: DosLingos 提供4个数据文件，在texts/目录下：

* news\_vocab\_sorted.txt - DosLingos 用户的词汇表(有序)
* scotttweet.txt - 上述Scott Harbaugh的推文
* twitter\_1k.txt - 用来测试效率较低的算法的1000条推文的小集合.
* twitter\_200k.txt - 用来测试效率较高的算法的200k条推文的大集合.

**你的工具**: DosLingos 已经有了读取文本文件的C0库，作为lib/readfile.c0提供给你，其中定义了类型string\_bundle并有以下函数：

// 首先，调用read\_words读取文件内容

string\_bundle read\_words(string filename)

你不用理解string\_bundle这个类型，但要知道可以从中取出它名下的字符串数组及其长度：

//从string\_bundle获取数组长度

int string\_bundle\_length(string\_bundle sb)

//获取string\_bundle中的数组

string[] string\_bundle\_array(string\_bundle sb)

//@ensures \length(\result) == string\_bundle\_length(wl);

下面是在Scott Harbaugh的推文上利用这些函数的例子：

$ coin lib/readfile.c0

--> string\_bundle bund = read\_words("texts/scotttweet.txt");

bund is 0xFFAFB8E0 (struct fat\_string\_array\*)

--> string\_bundle\_length(bund);

6 (int)

--> string[] tweet = string\_bundle\_array(bund);

tweet is 0xFFAFB670 (string[] with 6 elements)

--> tweet[0];

"phil" (string)

--> tweet[5];

"burrow" (string)

善用高效算法。DosLingos已经实现了他们自己的一套字符串搜索算法，在lib/stringsearch.c0中。或许你会发现在本任务中有用：

// Linear search

int linsearch(string x, string[] A, int n)

//@requires 0 <= n && n <= \length(A);

//@requires is\_sorted(A, 0, n);

/\*@ensures (-1 == \result && !is\_in(x, A, 0, n))

|| ((0 <= \result && \result < n)

&& string\_equal(A[\result], x)); @\*/

// Binary search

int binsearch(string x, string[] A, int n)

//@requires 0 <= n && n <= \length(A);

//@requires is\_sorted(A, 0, n);

/\*@ensures (-1 == \result && !is\_in(x, A, 0, n))

|| ((0 <= \result && \result < n)

&& string\_equal(A[\result], x)); @\*/

本练习的代码应该被放在文件doslingos.c0中。每个函数都要设置注释，包括前置条件(requires)，后置条件(ensures)和循环不变量(loop\_invariant)，也可以有一些必要的断言(assertation)。你可以把任何你需要的辅助功能函数放在文件中，但不能有main函数。可以在代码中包含lib/readfile.c0和lib/stringsearch.c0库中的功能：README.txt中的汇编指令包括了这些库。

你的测试应该放在文件doslingos-test.c0中，该文件中有一个main（）函数。Autograder(自动评分器)会检验你的测试，但是这些测试不会被评分。

## Task4（5分）创建一个文件doslingos.c0，文件包括按照下面的声明(declaration)写的一个函数count\_vocab

int count\_vocab(string[] vocab, int[] freq, int v,

string tweetfile,

bool fast)

//@requires v == \length(vocab) && v == \length(freq);

//@requires is\_sorted(vocab, 0, v) && all\_distinct(vocab, v);

该函数应返回文件tweetfile中未出现在数组vocab中的单词的出现次数，并应根据词汇表中每个单词出现的次数更新freq中的计数。如果一个单词在tweetfile中出现了多次，您应该分别计算每次出现的次数，因此推文“ha ha ha LOL LOL”将使“ha”的频率计数增加3；假设LOL不在词汇表中，则还将返回2。如果此增加(addition)导致溢出(overflow)，则不应该算作错误(error)。

需要注意，count\_vocab的前提是词汇表vocab必须有序，你应该充分利用这点。当fast的值是false时，你的函数应使用线性搜索算法；fast是true时，应该使用二分搜索算法。您可以用一个简单的if语句来决定调用哪个函数——复制大量的代码是不必要且无益的。

# 第三大题 单元测试

DosLingos公司原来的选择排序算法已经不能满足从大型文本中整理出词汇表的工作需求。你的同事们目前使用两个排序函数，都在sort.c0中提供：

void sort(string[] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

//@ensures is\_sorted(A, lower, upper);

string[] sortcopy(string[] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

//@ensures is\_sorted(\result, 0, \length(\result));

第一个函数属于就地排序，就像我们在课堂上讨论的，而第二个函数属于拷贝的类型，意为保持原数组A不变，并返回一个有序的、长度为upper-lower的数组（注意,这两个条件都还没有直接表达在约定中）。

DosLingos 决定付钱让另一家公司去写一个更快的、有相同接口的排序算法。不幸的是，他们没有意识到其他公司是闭源的，所以现在 你的公司的未来依靠着你看不见的代码，你知道程序是正确的，但是关于内部如何执行则一无所知。这至少导致了两个严重问题：

第一，你不能够证明你的排序函数总是遵行他们的约定——最多能给出一个反例，写一个可以使后置条件@ensures错误的测试。如果外部的承包公司给你这样一个完全虚假的函数实现(implementation)：

void sort(string[] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

//@ensures is\_sorted(A, lower, upper);

{

return;

}

你可以写一个含有main的，功能是排序的测试文件，来显示该函数出错了，并且观察到添加-d来编译执行时，后置条件@ensures不被满足。

第二，即使代码程序确实满足了约定，也不一定是正确的。比如，这个sortcopy函数永远满足后置条件，但它一定是错的：

string[] sortcopy(string[] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

//@ensures is\_sorted(\result, 0, \length(\result));

{

return alloc\_array(string, upper-lower);

}

为了捕捉这种不正确的排序算法，你必须写一个带有main()函数的测试文件，然后使用断言(assertation)检查其他保证正确性的条件——为了抓住这个bug，is\_in() 和/或 binsearch()两个函数可能有所帮助，虽然他们肯定是没有必要的(译者注：对排序实现本身来说)。你最后可能会倾向于在这项任务中使用//@assert ...;，而不是assert（...）;因为前者允许使用\length(A)来检查数组的长度。

为了得到下一个任务的全部分数，你需要编写额外的断言（assertation）,在外部承包商写了会不满足约定的函数或者满足约定但仍然错误的函数时(译者注：分别对应以上两个情况),断言都会显示error。

## 任务5（5分）写2个文件，sort-test.c0 and sortcopy-test.c0,，测试两个排序功能。

Autograder的评分根据你的单元测试是否能通过正确的sort样例(pass)、不允许各种有错的sort样例通过(fail)。你的测试仍然需要是安全的：当打开-d的时候，不允许进行数组的越界访问。

你并不需要捕捉所有的错误来得到满分，但捕捉到更多错误样例将反映在记分板上（作为附加分数的依据）。

因为你只能通过autograder访问所有错误样例，所以这项作业会在你提交后立马评分。

## 3.1 测试你的test.c0代码

通过运行以下命令，你可以用DosLingos公司自己的（正确的）选择排序算法的测试你写的函数，也能用本节给出的两个极其糟糕的函数来测试：

% cc0 -d -w lib/\*.c0 sort.c0 sort-test.c0

% ./a.out

% cc0 -d -w lib/\*.c0 sort.c0 sortcopy-test.c0

% ./a.out

% cc0 -d -w lib/\*.c0 sort-awful.c0 sort-test.c0

% ./a.out

% cc0 -d -w lib/\*.c0 sortcopy-awful.c0 sortcopy-test.c0

% ./a.out

这四个测试都应该成功编译并运行，但./a.out的最后两个调用应触发违反约定（如果你的测试在最低限度之上进行了完善的话）。我们一次只测试一个函数，所以sort-test.c0只能引用sort()函数, 而sortcopy-test.c0只能引用sortcopy()函数。两者都可以引用is\_sorted这个特定函数，和所有定义在lib/arrayutil.c0和lib/stringsearch.c0里的其他函数。

# 附录A 字符串处理概述

在C0这门语言中，一个字符串（string）是一个字符的序列。和C这样的语言不同的是，一个字符串（string）和一个字符数组（char []）并不相同。这里是字符串函数的一个库，你可以使用它来处理字符串string（通过#use <string>来在你的代码中使用这个库）。

// 返回字符串长度

int string\_length(string s)

// 返回字符串在给定下标位置上的字符

// 若下标超出范围，中止执行

char string\_charat(string s, int idx)

//@requires 0 <= idx && idx < string\_length(s);

// 返回由字符串a和b连接成的新串

string string\_join(string a, string b)

//@ensures string\_length(\result) == string\_length(a) + string\_length(b);

// 返回由字符串s的字符构成的子串[start, end)

// 若end<=start, 返回空串(译者注：根据下方的约定，这里应该是end==start返回空串，end<start会中止执行)

string string\_sub(string a, int start, int end)

//@requires 0 <= start && start <= end && end <= string\_length(a);

//@ensures string\_length(\result) == end - start;

bool string\_equal(string a, string b)

int string\_compare(string a, string b)

//@ensures -1 <= \result && \result <= 1;

// 由整型变量(ints)构建字符串

string string\_fromint(int i)

// 由布尔型变量(bools)构建字符串

string string\_frombool(bool b)

//由字符型变量(chars)构建字符串

string string\_fromchar(char c)

//@requires c != '\0';

//@ensures string\_length(\result) == 1;



图1 ASCII码对应表

string\_compare函数执行两字符串的字典序比较；字典序比较原先是一种用于字典的排序方法。单字符比较是基于ASCII码的，不止是基于字母顺序。因此，该顺序就叫做ASCII表顺序。ASCII表如图1。0的ASCII码为0x30（十进制48），A的ASCII码是0x41（十进制65），a的ASCII码是0x61（十进制97），注意到字符表这样的编排使得A小于B，B小于C，并以此类推。因此，ASCII表顺序与传统的字母表顺序大致相符。