

Simple Manual for Understanding the Background
of the Test Program for Project 4.
Anya Kim

SECTION 1. What is the frequency of the letters of the alphabet in English?

The inventor of Morse code, Samuel Morse (1791-1872), needed to know this so that he could give the simplest codes to the most frequently used letters.

The following table lists the characters of the English alphabet in alphabetical order, showing the frequency of the character's occurrence, and its rank based on that frequency. As you may have already guessed, 'e' is the most frequent, and 'z' is the least frequently used character.

Table 1. Frequency Table for the English Alphabet

Character	Frequency (%)	Rank
A	7.49	3
B	1.29	20
C	3.54	12
D	3.62	10
E	14.00	1
F	2.18	16
G	1.74	17
H	4.22	9
I	6.65	7
J	0.27	24
K	0.47	22
L	3.57	11
M	3.39	13
N	6.74	6
O	7.37	4
P	2.43	15
Q	0.26	25
R	6.14	8
S	6.95	5
T	9.85	2
U	3.00	14
V	1.16	21
W	1.69	18
X	0.28	23
Y	1.64	19
Z	0.04	26

Now, let's sort the above table by the frequency of the character. The result is shown in Table 2.

Table 2. Characters in the English Alphabet Sorted by Frequency

Character	Frequency (%)	Rank
E	14.00	1
T	9.85	2
A	7.49	3
O	7.37	4
S	6.95	5
N	6.74	6
I	6.65	7
R	6.14	8
H	4.22	9
D	3.62	10
L	3.57	11
C	3.54	12
M	3.39	13
U	3.00	14
P	2.43	15
F	2.18	16
G	1.74	17
W	1.69	18
Y	1.64	19
B	1.29	20
V	1.16	21
K	0.47	22
X	0.28	23
J	0.27	24
Q	0.26	25
Z	0.04	26

SECTION 2. Applications of Character Frequency

Knowledge of the frequency may have been useful for Morse, but why should it matter to us?

As a computer science student, you should learn about Huffman codes. Very simply put, Huffman coding is a data compression technique based on replacing strings that appear most frequently by shorter strings and strings that appear less frequently by longer strings. Huffman coding uses priority queues and binary trees. These concepts will be covered in the latter part of this course, so hopefully you will see a real world application for all these data structures that you have been learning.

Another computer science problem that utilizes frequency is the problem of solving encrypted (encoded, scrambled) messages. This is called a frequency analysis. A great example of its use is in Sir Arthur Conan Doyle's Sherlock Holmes series "The Return of Sherlock Holmes (1905) – The Adventure of the Dancing Men". In this story, Holmes successfully deciphers a hieroglyphic message by using his knowledge of frequencies of the English alphabet. Most of Sir Arthur Conan Doyle's works are

now online so if you've missed this one, I suggest you go read it, just for fun. One of the sites the story is posted at is:
<http://www.citsoft.com/holmes3.html>

Lastly, the frequency table is useful if you ever plan to be on the Wheel of Fortune and want to win a lot of money and prizes. Have you ever wondered why so many contestants initially guess 's' and 't'? Look at Table 2: 's' and 't' are the most frequently used consonants of the English language, so there's a good chance they'll be part of the puzzle.

SECTION 3. Using the Frequency Table in the Test Plan: Test_Single.adb

Now that we know how useful an alphabet frequency table is, let's actually use it.

What would be the appropriate data structure for the Frequency Table?
 To represent one character with its frequency and rank, we can use a record that contains three fields:

```

TYPE Alphabet is RECORD
    Name: Character;
    Freq: Float;
    Rank: LetterRange; --Letter range is an Integer Subtype
END RECORD;

```

```

TYPE AlphabetArray IS ARRAY(1..26) OF Alphabet;

```

To be able to quickly insert and delete from the linked list, we chose to first put each character record into an array: an array of record. The array is static – its values do not change once we initialize it, and that gives us a convenient way to insert, delete, and retrieve from our linked list.

The array will look like this

<i>index</i>	1	2	3	4	5	6	7	8	24	25	26
name	a	b	c	d	e	f	g	h	x	y	z
freq	7.49	1.29	3.54	3.62	14.0	2.18	1.74	4.22	0.28	1.64	0.04
rank	3	20	12	10	1	16	17	9	23	19	26

Looking at the package spec (Singly_Linked_Lists_Generic.ads), we can tell it is generic. In fact, there are several generic elements in it that we as the client should instantiate.

These generic elements are:

```

GENERIC
    InfoType
    KeyType
    "<"
    RetrieveKey
    DisplayInfo

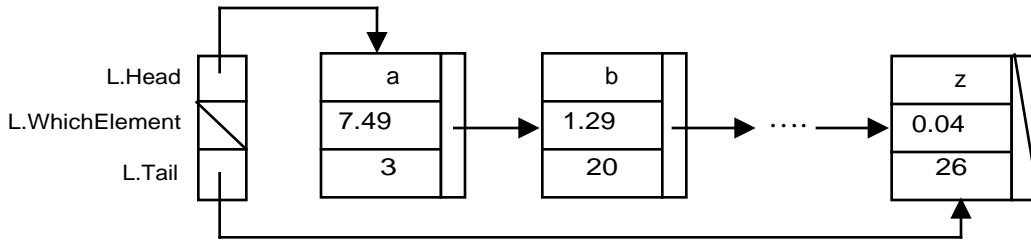
```

Lets talk about each in turn.

InfoType

We have to decide what our InfoType will be. Since we are trying to put the alphabet into a linked list and InfoType is the NodeType, we need the record type to be the InfoType so that in our client program, our linked list will look like:

Figure 1. Representation of Alphabet Records as a Linked List

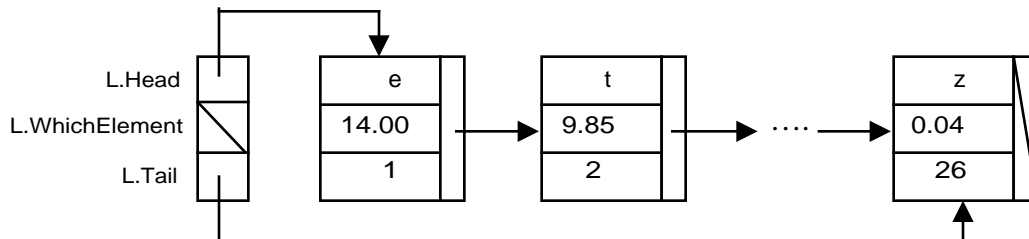


So we have decided that our InfoType will be the Alphabet Record.

KeyType

Looking at the package specification, we can guess that the KeyType, whatever it is, is used to sort the linked list. In Figure 1, the linked list seems to be sorted in alphabetical order. That is, it is sorted in order of the Name field of the Alphabet Record (`Alphabet.Name`). However, we decide that we think the list would be more use if it were sorted by decreasing value of frequency (such as shown in Table 2). If we represented this as a linked list, it would look like Figure 2:

Figure 2. Representation of Alphabet Records as a Linked List Sorted by Frequency



In order to sort by frequency, what would the key have to be? It can be either the `Freq` field or the `Rank` field of the Alphabet Record (`Alphabet.Freq` or `Alphabet.Rank`). However, notice that Rank increases while Frequency decreases. It seems more intuitive to use the Rank to me, so this test program will use the Rank field to order the linked list. Therefore, our KeyType in this case is an Integer. Note here that although `Alphabet.Rank` is declared as Type `LetterRange` (a subtype of `Integer`), we will not restrict KeyType to be `LetterRange`. The reason for this will be obvious as you look at the body of the test program, but I'll explain anyway. In some

of the tests, we try to delete and retrieve elements that are not in the given range. Restricting the KeyType to LetterRange (Integer values 1..26) would raise a constraint_error. We avoid this by instantiating KeyType to Integer.

“<”

Again, looking at the package specification tells us that this is some type of compare function that lets us compare the values of the KeyTypes. This function determines whether we will sort in ascending order or descending order. Since we want to imitate Table 2 (most frequently used characters are listed first: descending order of rank), we want this function to be a ‘less than’ function. Therefore, we will instantiate “<” to be “<”.

However, if we wanted to make our test program sort by ascending order of rank, all we would need to do is instantiate “<” to be “>”.... Even though its slightly confusing that way.

RetrieveKey

Since InfoType and KeyType are Generic, we need to tell the package how to extract the KeyType from the InfoType, so it can compare the values of two or more KeyTypes. Since we just defined InfoType and KeyType above, we know that in our client program, KeyType is a field of the InfoType. So it's a simple step to retrieve it.

DisplayInfo

This procedure should display InfoType. However, I hope you realize that being generic, its really up to you how much of the InfoType you want to display. For example, you could just display the Name and Freq fields of the Alphabet Record, or even just the Name field. Since this is a test program, we are going to display all the fields. Therefore we wrote in our test program, a Procedure DisplayOneElement that displays each field of the Alphabet Record with some formatting. Now all we need to do is instantiate DisplayInfo to be DisplayOneElement.

Putting It All Together

Now that we know what the generic components will be instantiated as in the client program, let's give this information to the package, too.

```
PACKAGE AlphaFreqList IS
    NEW Singly_Linked_lists_Generic (InfoType => Alphabet,
                                     KeyType => Integer,
                                     "<" => "<",
                                     RetrieveKey => GetRank,
                                     DisplayInfo => DisplayOneElement);
USE AlphaFreqList;
```

The USE clause is needed so we can omit the package name in front of data types, procedures and functions that belong to the package and are called in the client program.