Lab Assignment 0.

Eng 4892. MUN. Theodore Norvell, 2004

Due Thurs May 13. 11:00 PM.

Programming assignments will be marked according to the following scheme

Correctness	50%
Sensible naming and layout	10%
Sensible use of subroutines	10%
Quality of documentation .	15%
Simplicity of algorithm	15%

0 Run-length encoding

Images may be represented by sequences of integers. Each position in the sequence represents a different pixel position on the screen and each integer represents a different colour. For example a four by four image

red	red	red	red
red	blue	blue	blue
red	blue	blue	blue
red	red	red	red

might be represented by a sequence

$$\langle 31744, 31744, 31744, 31744, 31744, 31, 31, 31, 31744, 31, 31, 31, 31744, 31744, 31744, 31744 \rangle$$

where 31744 is the code for red and 31 is the code for blue. To send data like this over a network, it is often compressed. One compression technique is called run-length encoding. The idea is to create a list that contains colour values followed by numbers indicating how many repetitions of that colour value occur in the original sequence. Thus our sequence above would be encoded as a sequence

$$\langle 31744, 5, 31, 3, 31744, 1, 31, 3, 31744, 4 \rangle$$

meaning there are 5 repetitions of 31744, followed by 3 repetitions of 31, followed by 1 repetition of 31744, followed by 3 repetitions of 31744, followed by 4 repetitions of 314744. If colours take 2 bytes to represent and the repetition numbers are limited to 1 byte, then the length of the message that must be sent is reduced from 32 bytes to 15 bytes.

For this assignment you will *design and code* two subroutines using the "List" ADT discussed in class and a C++ implementation of it that I will provide for you.

Subroutine encode will take a list of integers and change it to a run-length encoded version. The repetition numbers should be in the range of 1 to 255 (since we want to send them as 1 byte) so you would encode 500 forty-twos as

$$\langle ..., 42, 255, 42, 245, ... \rangle$$

Within this constraint, the final list should be as short as possible.

Subroutine decode will take a compressed list and decompress it.

Your subroutines should have the following interface:

```
void encode( List<int> &text, bool &success ) ;
void decode( List<int> &text, bool &success ) ;
```

In both cases the success flag should be set to true if the subroutine successfully completes its task, and to false if it does not. The only reason that your encode subroutine may not complete their task is if an exception is thrown from a List<int> object. For decode, success should be set to false if an exception is thrown from a list object, but also if it is found that the input (the initial value of text) could not be the output of encode.

I will provide all the files you need on the web.

You should submit a file called `runlen.cpp`, using websubmit, as assignment 0.

1 N-way merge

For a number of data processing applications, you are faced with the following problem: You have a number of lists, each of which is sorted.[0] You want one list that contains all the data on all the input list and which is itself sorted. For example, if the input to the N-way merge is the 3 lists

⟨"eggs", "pancakes", "spam"⟩
⟨"bread", "spam", "spam"⟩
⟨"caviar", "spam"⟩

[0]For the purpose of this assignment 'sorted' means sorted according to the value returned by the 'strcmp' routine in the cstring include file.

then the output is

⟨“bread”, “caviar”, “eggs”, “pancakes”, “spam”, “spam”, “spam”, “spam”⟩

You are to write a subroutine

```
void merge(  List<List<char *> > inLists,
             List<char *> &outList,
             bool &success ) ;
```

which merges all the lists in inLists and places the result in outList. You may assume, as a precondition, that outList is initially empty. (Check this precondition with a call to the assert macro.) You may also assume, as a precodition, that each list within inLists is sorted. (You may check this precondition with calls to the assert macro, if you wish.) You may not assume that inList is not initially empty. The success flag must be set to true, if no exceptions are thrown, and false, otherwise.

Use ‘C style strings’. Remember that the include file cstring contains the declaration of strcmp, which may be used to compare strings. Don’t copy any characters, just the pointers.

For efficiency’s sake, I will add a method

```
T& retrieveRef( int i )
```

to the list ADT. You sould use this method to access lists within inList. E.g.

```
inList.retrieveRef(i).retrieve(j)
```

I will provide all the files you need on the web.

You should submit a file called `merge.cpp`, using websubmit command, as assignment 0.