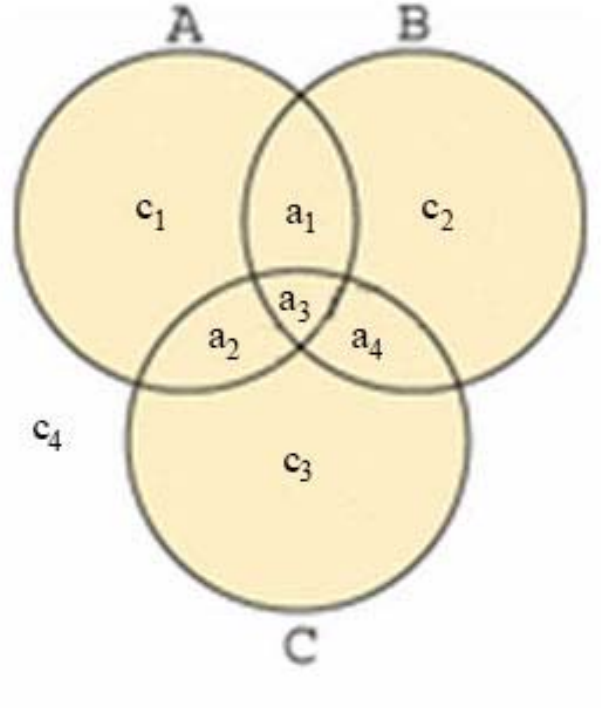
CS7A Lab 13 Binary Linear Codes

This lab is designed to give you practice using primitive data types together with arrays and strings. The application is to information/communication theory in the area of error detection and correction by the means of check digits.

Definition: A check digit is a number used to detect an error in data.

A very simple scheme (which was thoroughly explained in lecture, as if everyone was listening attentively). The idea is based on the Venn diagram showing the 8 regions where three sets A, B, C can intersect:



Think of the message as being encoded as the nibble of binary digits $a_1a_2a_3a_4$ of which there are 16: $0000, 0001, 0010, 0011, 0100, \ldots, 1111$. To build redundancy into the message sending/receiving system (which may include "noise" or errors in transmission) we add a nibble of check digits c_1, c_2, c_3, c_4 as illustrated above, with the idea the sum of digits in A, B, C and all together is even. That is,

$$(a_1 + a_2 + a_3 + c_1)\%2 == 0$$

$$(a_1 + a_3 + a_4 + c_2)\%2 == 0$$

$$(a_2 + a_3 + a_4 + c_3)\%2 == 0$$

$$(a_1 + a_2 + a_3 + a_4 + c_1 + c_2 + c_3 + c_4)\%2 == 0$$

The process of sending the code with the check digits then involves

1. Encode the message as a sequence of nibbles by converting the ascii characters of the message into binary (the is the `convert2bin()` function outlined below.)
2. Simulate the transmission of the information from a sender to a receiver over a noisy channel by adding "noise" (errors) to the encoded message and the check digits. Provide for various levels of noise to see how well the check digits work.
3. Decode the message on the receivers end by converting the binary data back to ascii values. Use the check digits to detect errors in the transmission.
4. Can you use the check digits to correct errors (this is above and beyond the basic requirements of this lab.)

Here is a starter shell for this lab:

```
 // G. Hagopian
// Implementing error detection/correction scheme for a binary linear code
/* codewords:
11111110
11101001
11010001
10110101
01110011
11000110
10100010
10011010
01100100
01011000
00111000
00010111
00101111
01001011
10001101
00000000
*/
#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#define SIZE 128
using namespace std;

void convert2bin(char*, bool b[], bool c[]);
void showBin(bool b[]);
void addNoise(bool b[], bool c[], int);
void decode(bool [], char*);
int detect(bool b[], bool c[]);
void correct(char*, bool[], bool c[]);

int main()
{
      bool binWord[SIZE] = {0}, binCzek[SIZE] ={0};
      int noise;
```

```
	char word[5] = "AAAA";
	//get a message
	cout << "\nEnter a four-letter word: ";
	cin >> word;
	cout << "\nYou entered " << word << endl;
	cout << "\nIn binary, that's ";
	convert2bin(word, binWord, binCzek);
	showBin(binWord);
	showBin(binCzek);
	cout << "\nEnter a noise level (digit between 1 and 4)";
	cin >> noise;
	decode(binWord, word);
	cout << "\nBefore adding noise, we decode the binWord as " << word;
	addNoise(binWord, binCzek, noise);
	cout << "\nAfter the noise, the data are " << endl;
	showBin(binWord); showBin(binCzek);
	decode(binWord, word);
	cout << "\nAfter adding noise, we decode the binWord as " << word;
	cout << "\nWe detect " << detect(binWord, binCzek)
	     << " errors." << endl;
}

void convert2bin(char* w, bool b[], bool c[])
{
	/*Each ascii number is converted binary in the array b,
	then the check digits for these are stored in the array c.*/
	int temp;
	for(int i = 0; i < 5; i++)  // for each letter
	{
		temp = w[i];
		//cout << "\ntemp = " << temp;
		for(int j = 8*(i+1)-1; j >= 8*i; --j) //convert to binary
		{
			b[j] = temp%2;
			temp /= 2;
		}
		// do redundancy
		c[8*i] = ((int)b[8*i]+(int)b[8*i+1]+(int)b[8*i+2])%2;
		c[8*i+1] = ((int)b[8*i]+(int)b[8*i+2]+(int)b[8*i+3])%2;
		c[8*i+2] = ((int)b[8*i+1]+(int)b[8*i+2]+(int)b[8*i+3])%2;
		c[8*i+3] = ((int)b[8*i]+(int)b[8*i+1]+(int)b[8*i+2]+(int)b[8*i+3]
				+(int)c[8*i]+(int)c[8*i+1]+(int)c[8*i+2])%2;
		c[8*i+4] = (b[8*i+4]+b[8*i+5]+b[8*i+6])%2;
		c[8*i+5] = (b[8*i+4]+b[8*i+6]+b[8*i+7])%2;
		c[8*i+6] = (b[8*i+5]+b[8*i+6]+b[8*i+7])%2;
		c[8*i+7] = (b[8*i+4]+b[8*i+5]+b[8*i+6]+b[8*i+7]
				+c[8*i+4]+c[8*i+5]+c[8*i+6])%2;
	}
	cout << endl;
}

void showBin(bool b[])
{
	for(int i = 0; i < SIZE/4; i++)
	{
		cout << b[i];
		if((i+1)%8==0) cout << " ";
```

```
	}
	cout << endl;
}

void addNoise(bool b[], bool c[], int noiseLevel)
{
	//define addNoise here
	int currentPos = 0;
	while(currentPos < 32)
	{
		currentPos += rand()%(32/noiseLevel);
		b[currentPos] += 1;
	}
	currentPos = 0;
	while(currentPos < 32)
	{
		currentPos += rand()%(32/noiseLevel);
		c[currentPos] += 1;
	}
}

void decode(bool b[], char*)
{ //define decode here
}

int detect(bool b[], bool c[]){
	//define detect here
	return 0;
}

void correct(char*, bool[], bool c[]){
	//define correct here
}
```