Project 1- Hamming code
Submitted by Deepak Shah
Hardware Design 241
To: Professor Brown

Attached is my codes for Hamming code Project. During this project, I read some articles to better understand ways to convert a hexadecimal number into bits.

Below are the sources I read:

1. http://blog.refu.co/?p=804
2. http://www.evanmiller.org/four-days-of-go.html

After understanding the main idea, I was able to use my own previous codes to remind myself of ways to approach the problem in a more efficient way.

✓ I pledge my honor that I have neither given nor received assistance on this programming project, except as explicitly stated on this page, and that I have seen no dishonest work.

Signed: Deepak Shah

```
//Project 1 - This code gives two options to users. Option 1 computes the hamming code and
Option 2 checks the input hammingcode and prints the correct hammingcode, if wrong.
/* Deepak Shah Hammingcode Project*/

#include<iostream>
#include<math.h>
using namespace std;

int bit[21];//An array of containing the bits of the hexadecimal code. All these bits then get
transferred to hcode array on a specific pattern.
int hcode[21];//An array of length 21 containing bits of the hexadecimal code. This array has
bits arranged with parity bits at 1,2,4,8,16th position and data bits at rest
int checkbit[5];//An array of length 5 containing integers.
int hambit[5];//An array of length 5 containing integers.
int hexd[5];//An array of length 5 containing integers.
int x,hx;
int* correctcode();
/*Function to store all the bits in an array of int*/
int* storeinarray(int length){
	bit[(length-1)]=(x & 1);//The least significant bit is stored in the last address of
the array.
	for(int i=1;i<length;i++){
		bit[(length-1)-i]=(((x) & (1<<i))>>i); //Most important part of the code. This
code takes the least significant bit one after another and then stores that bit in an array of
int. The right most bit is stored in the rightmost part of array.
	}
	return bit;
}
/*Function to create another int array of length 21 and transfer the 16 bits in appropriate
order*/
int* storehamarray(){
	int j=0;
	for(int i=0;i<21;i++){//loop occurs for 17 times. Does not occur when the value of i
satisfies the values mentioned in the If condition.
		if(i!=0 and i!=1 and i!=3 and i!=7 and i!=15){
			hcode[i]=bit[j];//if condition met, then values from bit array are
copied to hcode array.
			j++;//increment of j
		}
	}
	return hcode;//returns the array containing all the bits
}
/*This function actually copies all the bits from bit array to hcode array. This function is
not used by the Case 1 (Hammingcode Conversion) but used by Case 2(Correction of
Hammingcode)*/
int* copypaste(){
	for(int i=0;i<21;i++){

		hcode[i]=bit[i];
```

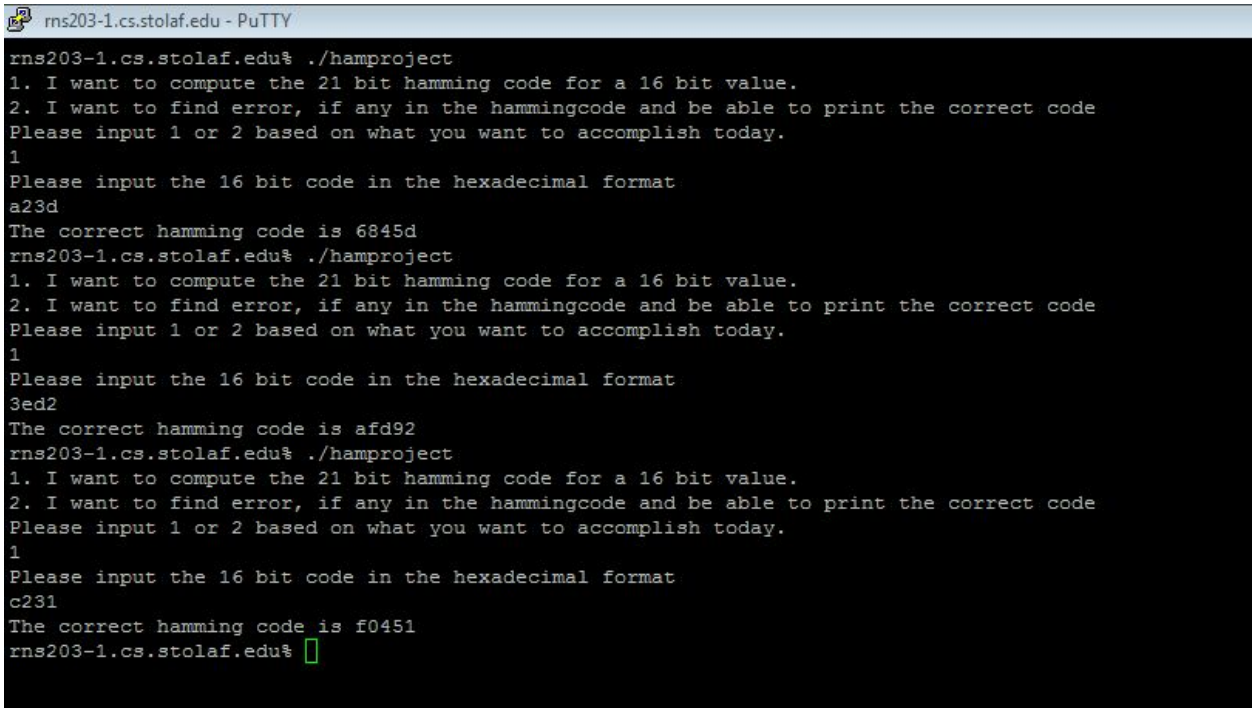
```
	}
	return hcode;
}
/*Function to compute the checkbits based on the sum of the data bits at position 1,2,4,8 and 16.*/
void hammingcode(){


checkbit[0]=hcode[2]+hcode[4]+hcode[6]+hcode[8]+hcode[10]+hcode[12]+hcode[14]+hcode[16]+hcode[18]+hcode[20];

checkbit[1]=hcode[2]+hcode[5]+hcode[6]+hcode[9]+hcode[10]+hcode[13]+hcode[14]+hcode[17]+hcode[18];

checkbit[2]=hcode[4]+hcode[5]+hcode[6]+hcode[11]+hcode[12]+hcode[13]+hcode[14]+hcode[19]+hcode[20];
	checkbit[3]=hcode[8]+hcode[9]+hcode[10]+hcode[11]+hcode[12]+hcode[13]+hcode[14];
	checkbit[4]=hcode[16]+hcode[17]+hcode[18]+hcode[19]+hcode[20];
	for(int i=0;i<5;i++){
		if(checkbit[i]%2==0)
			hambit[i]=0;
		else hambit[i]=1;

	}


}
/*//Function to check if all the 21 bits in the code are correct. This function is only required in Case 2 (Correcing Hammingcode). This function returns a true if there is no error in the hammingcode and false otherwise.*/
bool checkhammingcode(){

	int c=0;//A variable that whose value increases if there is an error in the calculated bits in the code with that of the bits stored in the array.

	for(int i=0;i<5;i++){
		int z=0;
		z=(pow(2,i)-1);//pow(2,i) returns 2^i. In this case, bits 0,1,3,7,15 all can be expressed mathematically by a formula (2^i-1). This formula has been used in this loop to reach these bits.

		if(hambit[i]!=hcode[z]){
			c++;
		}
	}

	if (c>0){

		return false;
```

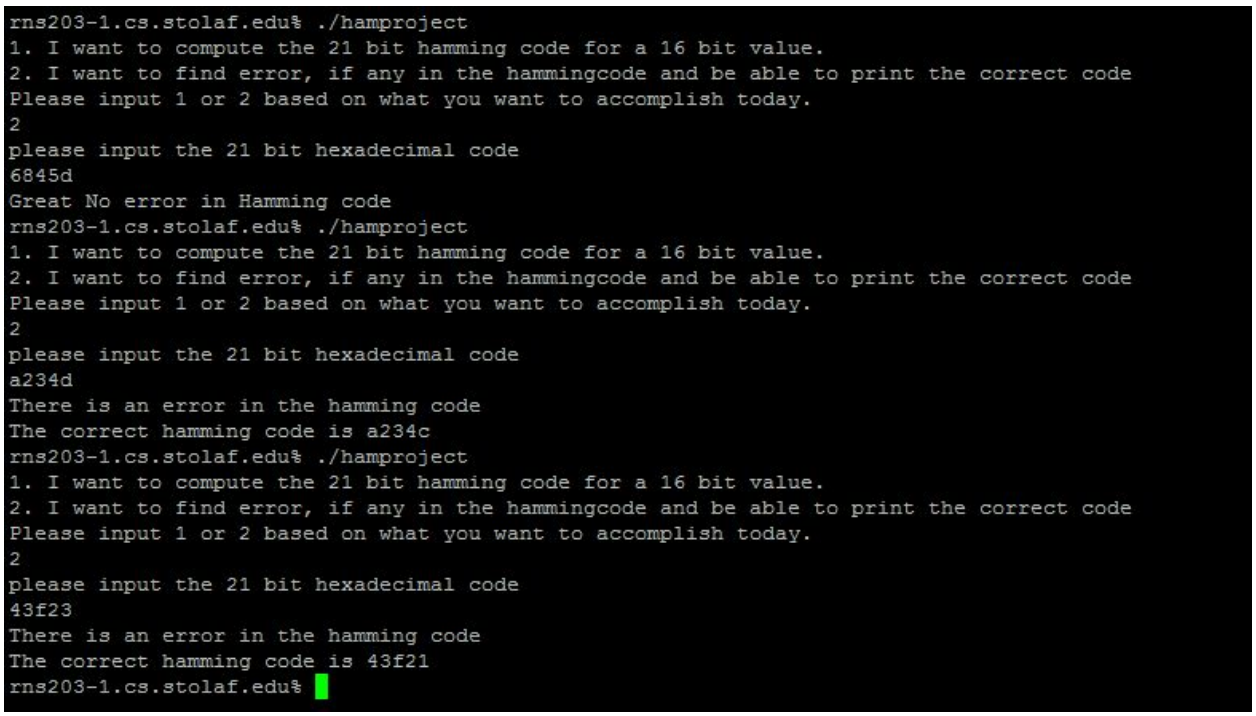
```
}
        else {

                return true;

        }



}
/*Function to add the checkbits in the 21 bit code*/
int* addhammingbits(){
        int z=0;

        for(int i=0;i<5;i++){
                z=(pow(2,i)-1);
        hcode[z]=hambit[i];
        }
        return hcode;

}

/*Function which changes the incorrect bits with the correct ones. This function is only used
in Case 2.*/
int* correctcode(){
int y=0,z=0;
int sum=0;
  for(int i=0;i<5;i++){
        z=(pow(2,i)-1);
        if(hcode[z]!=hambit[i]){
                sum=sum+pow(2,i);//This adds the incorrect bits positions and adds them to
compute the bit position which contains the incorrect bit.
                y++;
        }
  }
  if (y>0){
        if(hcode[sum]==0) hcode[sum]=1;
        if(hcode[sum]==1) hcode[sum]=0;

  }
  return hcode;


}
/*Function to print the binary into Hexadecimal format. Done Manually rather than using the
inbuilt function cout<<hex. This function stores the hexadeimal numbers in an int array and
then prints the array.*/
void bin2hex(){
        int decimal=0;
for(int i=0;i<21;i++){
```

```
        decimal+=(hcode[20-i]*pow(2,i));
}
   int i=0;
    while(decimal>0){
                hexd[i]=decimal%16;
                decimal=decimal/16;
                i++;
        }
 for (int i=4;i>=0;i--){
                if (hexd[i]==10)
                        hexd[i]='a';
                else if (hexd[i]==11)
                        hexd[i]='b';
                else if (hexd[i]==12)
                        hexd[i]='c';
                else if (hexd[i]==13)
                        hexd[i]='d';
                else if (hexd[i]==14)
                        hexd[i]='e';
                else if (hexd[i]==15)
                        hexd[i]='f';
                else hexd[i]=hexd[i];
        }
    cout<<showbase<<dec;
    cout<<"The correct hamming code is ";
        for (int i=4;i>=0;i--){
        if(hexd[i]>96 and hexd[i]<123)
                cout<<(char)hexd[i];//values like 10 to 15 are also stored in this array.
However the hexadecimal equivalence of these values are a to e, so (char) has been used to
return the the letters using the ascii values.
        else cout<<hexd[i];;
}
cout<<" "<<endl;

}
/*//Function to print the hamming code of an input 16 bit code*/
void hamming(int x){
        storeinarray(16);
        storehamarray();
    hammingcode();
    addhammingbits();
    bin2hex();
}
/*// Function that checks the 21 bit Hamming code and prints the correct code if wrong*/
void hamcheck(int x){

        storeinarray(21);
        copypaste();
        hammingcode();
        if (checkhammingcode()==true){
```

```
                cout<<"Great No error in Hamming code"<<endl;}
                else{
                        cout<<"There is an error in the hamming code"<<endl;
                        correctcode();
                        bin2hex();
                }

}

int main(){
int n;
cin.setf(ios::hex, ios::basefield);
cout.setf(ios::hex, ios::basefield);
cout.setf(ios::showbase);
cout<<"1. I want to compute the 21 bit hamming code for a 16 bit value. "<<endl;
cout<<"2. I want to find error, if any in the hammingcode and be able to print the correct
code"<<endl;
cout<<"Please input 1 or 2 based on what you want to accomplish today."<<endl;
cin>>n;
switch(n){
        case 1:
        cout<<"Please input the 16 bit code in the hexadecimal format"<<endl;
        cin>>x;
        hamming(x);
        break;
    case 2:
    cout<<"please input the 21 bit hexadecimal code"<<endl;
    cin>>x;
    hamcheck(x);
    break;
    default:
    cout<<"Please  enter only either 1 or 2"<<endl;
    break;
}

return 0;
}
```

Output from sample runs:





Extra Features: Apart from finding the hamming code of an input 16 bit code, this code also allows the user to check if the input 21 bit hamming code is correct or not. If the hamming code is incorrect, then it successfully prints the correct hamming code.