**STEPHEN BEKO – CODING TEST**

**Puzzle #1**) Write a program that prints all sequences of 32 digits, such that each digit in the sequence is a 0 or 1, and no sequence has two 1's adjacent in the output. For example, the following sequences should be included in the output (not necessarily in this order):

… 00000000000000000000000000000000

... 01010101010101010101010101010101

... 01000000010000000100000001000000 ...

But the following sequence should not be printed, because it has 1's adjacent in the output:

… 01100000011000000110000001100000

Program Sequence to allow and test against

00000000 0 00000001 1 00000010 2 00000100 4

00000101 5 00001000 8 00001001 9 00001010 10

00010000 16 00010001 17 00010010 18 00010100 20

00010101 21 00100000 32 00100001 33 00100010 34

00100100 36 00100101 37 00100100 40 00101001 41

00101010 42

**Coded and tested in vis studio**.

Change MAX\_BITS to 32 for full binary range 0 to 0xFFFFFFFF

**ANSWER**

#include <stdio.h>

#include <math.h>

#define MAX\_BITS 6 // CHANGE THIS TO 32 for full range

/\*

Function converts uint32 to string binary

\*/

void pf(unsigned int val, char \*buf)

{

int a, b;

for (a = 31; a >= 0; a--)

{

b = val >> a;

if (b & 0x1)

buf[31-a] = '1';

else

buf[31-a] = '0';

}

buf[32] = NULL;

}

int main()

{

int n; // bit counter 0 - 31

unsigned int bit\_start;// Start UL to test in range. 2^n

unsigned int bit\_end; // End UL to test for bit position, 2^(n+1) - 1

unsigned int num; // Bit pattern to test, initialise to next bit pos

int bit\_shift; // count of shift operations

bool print\_flag; // print this or not

bool bit\_set; // is bit a 1

bool bit\_prev; // is bit a 0

char buff[40]; // to print binary

// initialise

n = bit\_start = bit\_end = num = 0;

// while loop for each consecutive bit position to test

while (n < MAX\_BITS)

{

bit\_start = pow(2, n); //2 ^ n;

// special case when 0, otherwise algorithm works

if (n == 0)

bit\_start = 0;

bit\_end = pow(2, n+1) -1; //2 ^ (n + 1) – 1;

// test each number in range

for (num = bit\_start; num <= bit\_end; num++)

{

print\_flag = true;

bit\_shift = 0;

bit\_prev = false;

// test num bits for adjacent ‘1’s and print if not present

for (bit\_shift = 0; bit\_shift < (n + 1); bit\_shift++)

{

bit\_set = false;

if ((num >> bit\_shift) & 0x1)

{

bit\_set = true;

if (bit\_set && bit\_prev)

{

print\_flag = false;

break;

}

}

bit\_prev = bit\_set;

} //end for

// if we have adjacent ‘1’s don’t print otherwise print num

if (print\_flag)

{

pf(num, buff);

printf("0x%x %lu %s\n", num, num, buff);

}

} //end for

n++;

} //end while

return 0;

}

**Puzzle #2** --- Write a program to accept a C char\* string (or a ulong), that will print the offsets of runs of 0's and 1's, where the runs are longer than 1. For example, if given "0010011" it will print "0, 3, 5".

Position of ‘00’, ‘00’, ‘11’

**Coded and tested in vis studio**.

**ANSWER**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NUM\_RUNS 50

int main(int argc, char \*argv[])

{

int match\_pos[MAX\_NUM\_RUNS]; // position of run, set on second consecutive bit

int pos = 0; // index for above array

bool match; // match set - only set once per run

int cnt\_zero; // count '0s'

int cnt\_one; // count '1s'

int len; // len of string passed in

int i;

char \*cptr; // pointer to string

if (argc != 2) {

printf("program <binary string>");

exit(0);

}

// initialise

for (i = 0; i<10; i++)

{

match\_pos[i] = 0;

}

len = strlen(argv[1]);

cptr = argv[1];

match = false;

printf("string=%s len=%d\n", cptr, len);

cnt\_one = cnt\_zero = 0;

// loop to process ‘1s’ and ‘0s’

for (i = 0; i< len; i++)

{

if (pos > MAX\_NUM\_RUNS)

{

printf("unable to process further\n");

break;

}

if(\*cptr++ == '1')

{

cnt\_one++;

cnt\_zero = 0;

}

else

{

cnt\_zero++;

cnt\_one = 0;

}

if((cnt\_one >= 2) || (cnt\_zero >= 2))

{

if (!match)

{

// set once for first position

match\_pos[pos] = (i - 1);

match = true;

pos++;

}

}

else

{

match = false;

}

}

// now print it

printf("Runs of '0s' or '1s' are: ");

for (i = 0; i< pos; i++)

{

printf("%d ", match\_pos[i]);

}

}