// Step 3

////////////////////////////////////////////////////////////////////////////////

//

// Function : float\_display\_array

// Description : displays an array of floats

//

// Inputs : arr - float array

// n - number of elements

// Outputs : number of items in array

int float\_display\_array(float \*arr, int n){

// int n = sizeof(arr)/sizeof(arr[0]);

// printf("%d\n",n );

for(int i=0; i<n; i++){

printf("%f\n", arr[i]);

}

return n;

}

////////////////////////////////////////////////////////////////////////////////

//

// Function : integers\_display\_array

// Description : displays an array of integers

//

// Inputs : arr - integer array

// n - number of elements

// Outputs : number of items in array

int integer\_display\_array(int \*arr, int n){

// int n = sizeof(arr)/sizeof(arr[0]);

// printf("%d\n",n );

for(int i=0; i<n; i++){

printf("%d\n", arr[i]);

}

return n;

}

// step 4

int countBits(int integer){

int count = 0 ;

while (integer > 0){

if(integer & 1 ){

count++;

}

integer >>= 1 ;

}

return count;

}

// Step 5

////////////////////////////////////////////////////////////////////////////////

//

// Function : most\_values

// Description : Return the number of even numbers in the array (float version)

//

// Inputs : arr - float array

// n - number of items in array

//

//

// Outputs : count of even numbers in array.

int integerQuickSort(int \*arr,int l, int r){

int j;

if(l < r){

// devide

j = partition(arr, l, r);

integerQuickSort(arr, l, j-1);

integerQuickSort(arr, l+1, r);

}

}

void swap (int \*x, int \*y){

if (x != y){

\*x = \*x ^ \*y;

\*y = \*x ^ \*y;

\*x = \*x ^ \*y;

}

}

int partition( int a[], int l, int r) {

int pivot, i, j, t;

pivot = a[l];

i = l; j = r+1;

while( 1)

{

do ++i; while( a[i] <= pivot && i <= r );

do --j; while( a[j] > pivot );

if( i >= j ) break;

// swap(&a[i],&a[j]);

t = a[i];

a[i] = a[j];

a[j] = t;

}

// swap(&a[i],&a[j]);

t = a[l];

a[l] = a[j];

a[j] = t;

return j;

}

//Step 6

////////////////////////////////////////////////////////////////////////////////

//

// Function : most\_values

// Description : Return the number of even numbers in the array (float version)

//

// Inputs : arr - float array

// n - number of items in array

//

//

// Outputs : count of even numbers in array.

int float\_even(float \*arr, int n){

int temp = 0;

int count = 0;

for(int i=0;i<n; i++){

if (!((int) truncf(arr[i]) & 1 ))

count++;

}

return count;

}

////////////////////////////////////////////////////////////////////////////////

//

// Function : most\_values

// Description : Return the number of even numbers in the array (integer version)

//

// Inputs : arr - integer array

// n - number of items in array

//

//

// Outputs : count of even numbers in array.

int integer\_even(int \*arr, int n){

int temp = 0;

int count = 0;

for(int i=0;i<n; i++){

if (! (arr[i] & 1) )

count++;

}

return count;

}

////////////////////////////////////////////////////////////////////////////////

//

// Function : most\_values

// Description : Print out the values with the most occurences in array

//

// Inputs : arr - integer array

// n - number of items in array

// max - maximum possible value

//

// Outputs : value(s) that occurs the most.

int most\_values(int \*arr, int n, int max){

int counts[max];

int largest=0;

int countOfLargest;

int numberOfValues=1; // number of values for output

int i = 0;

for(i=0;i<max;i++) counts[i] = 0;

for(i=0;i<n; i++) {counts[arr[i]]=counts[arr[i]]+1;} // count each

for(i=0; i<max;i++){

if(counts[i] > largest)

largest = counts[i];

}

for (i=0; i<max;i++){

if (counts[i] == largest){

printf("%d\n",i);

}

}

return 0;

}

////////////////////////////////////////////////////////////////////////////////

//

// Function : binaryString

// Description : receives a pointer to a string, a length and a number to convert to binary.

//

// Inputs : str - string

// size - nusize of string

// number - number

//

// Outputs : prints

void binaryString(char \*str, int size, int number){

int i = size-2;

for(;i>=0;i--){

if(number & (1 << i)){

str[size-i-2] = '1';

}else{

str[size-i-2] = '0';

}

}

str[size-1] = '\0';

}

////////////////////////////////////////////////////////////////////////////////

//

// Function : reverseBits

// Description : revrses bit of the input

//

// Inputs : number - input unsigned short

//

//

// Outputs : unsigned short integer which is the reverse bits of input

unsigned short reverseBits(unsigned short number){

int size = 16;

int i = size-1;

unsigned short reverse = 0;

for(;i>=0;i--){

if(number & (1 << i)) // if ith bit is 1

reverse |= (1 << ((size-1) - i )) ;// set size - ith bit to zero

}

return reverse;

}