Memory leak vs Dangling pointer

Memory leak:
When there is a memory area in a heap but no variable in the stack pointing to that memory. ...
Dangling pointer:
When a pointer variable in a stack but no memory in heap. char *p =NULL; A dangling pointer trying to dereference without allocating space will result in a segmentation fault.

Is passing a pointer argument, pass by value in C++?

Pointers are passed by value as anything else. That means the contents of the pointer variable (the address of the object pointed to) is copied. That means that if you change the value of the pointer in the function body, that change will not be reflected in the external pointer that will still point to the old object. But you can change the value of the object pointed to.

If you want to reflect changes made to the pointer to the external pointer (make it point to something else), you need two levels of indirection (pointer to pointer). When calling functions it's done by putting a `&` before the name of the pointer. It is the standard C way of doing things.

When using C++, using references is preferred to pointer (henceforth also to pointer to pointer).

For the *why* references should be preferred to pointers, there is several reasons:

- references introduce less syntaxic noise than pointers in function body
- references keep more informations than pointers, than can be useful for compiler

Drawbacks of references are mostly:

- they break the simple pass-by-value rule of C, what makes understanding the behavior of a function regarding of parameters (will they be changed ?) less obvious. You also need function prototype to be sure. But that is not really worse than the multiple pointer levels necessary when using C.

- they are not supported by C, that can be a problem when you write code that should work with both C and C++ programs (but that's not the most usual case).

In the specific case of pointer to pointer, the difference is mostly simplicity, but using reference it may also be easy to remove both levels of pointers and pass only one reference instead of a pointer to pointer.

Size of array and pointer

Size of an array is the number of elements multiplied by the type of element, that is why we get sizeof arri as 12 and sizeof arrc as 3. Size of a pointer is fixed for a compiler. All pointer types take the same number of bytes for a compiler. That is why we get 4 for both ptri and ptrc.

Void pointer and NULL pointer

Null pointer is a special reserved value of a pointer. A pointer of any type has such a reserved value. ... Void pointer is a specific pointer type - void * - a pointer that points to some data location in storage, which doesn't have any specific type. So, once again, null pointer is a value, while void pointer is a type.

Pointer output questions

https://www.geeksforgeeks.org/c-language-2-gq/pointers-gq/

Mapping of 2d and 1d array
Call by value VS call by reference
Exception handling
Stacks
Queues
Fibonacci series
Palindrome string
Armstrong number
Prime number
Numbers swapping
Decimal to binary

Array logical questions

Remove duplicates from array with smallest missing value

Input []={0,1,2,1,3,2,3,5}
output[]={01,2,3,4,6,7,5}

```
 const int size=8;
    int arr[8]={0,1,2,1,3,2,3,5};
    int count[size]={0};
    //a frequency array to count frequency of each element and to find missing elements in array
    for(int i=0;i<size;i++){
        count[arr[i]]++;
    }

    int k=0;
    for(int i=0;i<size;i++){
        for(int j=i+1;j<size;j++){
            k=0;
            if(arr[i]==arr[j]){
                while(k<size){
                    if(count[k]==0){
                        arr[j]=k;
                        count[k]=1;

                        break;
                    }
                    k++;
                }



            }



        }
    }
    for(int i=0;i<size;i++){
        cout<<arr[i] <<" ";
    }
```

Find all repeating elements in array in one loop

```
void printDuplicates(int arr[], int n)
{
    int i;

    // Flag variable used to
    // represent whether repeating
    // element is found or not.
    int fl = 0;

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

        // Check if current element is
        // repeating or not. If it is
        // repeating then value will
        // be greater than or equal to n.
        if (arr[arr[i] % n] >= n) {

            // Check if it is first
            // repetition or not. If it is
            // first repetition then value
            // at index arr[i] is less than
            // 2*n. Print arr[i] if it is
            // first repetition.
            if (arr[arr[i] % n] < 2 * n) {
                cout << arr[i] % n << " ";
                fl = 1;
            }
        }

        // Add n to index arr[i] to mark
        // presence of arr[i] or to
        // mark repetition of arr[i].
        arr[arr[i] % n] += n;
    }

    // If flag variable is not set
    // then no repeating element is
    // found. So print -1.
    if (!fl)
        cout << "-1";
}
```

Print zeroes at the end of array

```
const int size=9;
   int arr[size]={0,1,2,0,0,3,0,5,0};
   int j=0;
   for(int i=0;i<size;i++){
      if(arr[i]!=0){
         arr[j++]=arr[i];

      }
   }
   while(j<size) {arr[j]=0;j++;}
   for(int i=0;i<size;i++) cout<<arr[i]<<" ";
```

Print zeroes at the end of array without counting zeros

```
void putZeroesAtEnd(int arr[], int size){
         int end_ptr = size - 1;
         int i = 0;
         while (i < size)
         {
                  if (i == end_ptr) break;
                  if (arr[i]==0 && arr[end_ptr]!=0)
                  {
                                    int temp = arr[i];
                                    arr[i] = arr[end_ptr];
                                    arr[end_ptr] = temp;
                                    end_ptr --;
                                    i++;

                  }
                  else if (arr[i] == 0 && arr[end_ptr] == 0)  End_ptr--;
                  else  i++;
         }
         for (int j = 0; j < size; j++)
         {
                  cout << arr[j] << " ";
         }
```

}

Find maximum times repeating element in sorted array in one loop

```
#include<iostream>
using namespace std;
void findMaxMode(int arr[], int size){
	int count = 1; int maxCount = 0;
	int element = 0;
	for (int i = 1; i < size; i++){
		if (arr[i] == arr[i - 1]){
			count++;
		}
		else {
			if (count>maxCount){
				maxCount = count;
				element = arr[i - 1];

			}
			count = 1;
			}

	}
	if (count > maxCount)
	{
		maxCount = count;
		element = arr[size - 1];
	}
	cout << element << " is repeating " << maxCount << " times\n";

}
int main(){
	int arr[] = {1,1,1,2,2,2,2};
	int size = 7;
	findMaxMode(arr, size);

}
```

Find maximum times repeating element in unsorted array

```
#include <iostream>
using namespace std;
void findMaxRepeating(int arr[],int size){
    int lar=arr[0];
    int ind;
    for(int i=0;i<size;i++){
        if(lar<arr[i]){
            lar=arr[i];
            ind=lar;

        }
    }
    const int count_size=lar+1;
    int count[count_size]={0};
    for(int j=0;j<size;j++){
        count[arr[j]]++;

    }
    lar=count[1];
     for(int i=1;i<count_size;i++){
        if(lar<count[i]){
            lar=count[i];

        }
    }
    cout<<ind<<" is repeating "<<lar<<" times\n";


}


int main(){

   const int size=6;
    int arr[size]={1,4,2,4,4,3};
    findMaxRepeating(arr,size);
}
```

Find smallest missing number in an unsorted array

```
int findMissingPositive(int arr[], int size)
{
    int i;

    // Mark arr[i] as visited by making arr[arr[i] - 1] negative.
    // Note that 1 is subtracted because index start
    // from 0 and positive numbers start from 1
    for (i = 0; i < size; i++) {
        if (abs(arr[i]) - 1 < size && arr[abs(arr[i]) - 1] > 0)
            arr[abs(arr[i]) - 1] = -arr[abs(arr[i]) - 1];
    }

    // Return the first index value at which is positive
    for (i = 0; i < size; i++)
        if (arr[i] > 0)
            // 1 is added because indexes start from 0
            return i + 1;

    return size + 1;
}
```

Put negative elements at the start and positive at end of array

```
void rearrange(int arr[], int n)
{
    // The following few lines are
    // similar to partition process
    // of QuickSort. The idea is to
    // consider 0 as pivot and
    // divide the array around it.
    int i = -1;
    for (int j = 0; j < n; j++)
    {
        if (arr[j] < 0)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
}
```

Put alternate positive and negative elements in array

```
void rearrange(int arr[], int n)
{
    // The following few lines are
    // similar to partition process
    // of QuickSort. The idea is to
    // consider 0 as pivot and
    // divide the array around it.
    int i = -1;
    for (int j = 0; j < n; j++)
    {
        if (arr[j] < 0)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    // Now all positive numbers are at
    // end and negative numbers at the
    // beginning of array. Initialize
    // indexes for starting point of
    // positive and negative numbers
    // to be swapped
    int pos = i + 1, neg = 0;

    // Increment the negative index by
    // 2 and positive index by 1,
    // i.e., swap every alternate negative
    // number with next positive number
    while (pos < n && neg < pos &&
                     arr[neg] < 0)
    {
        swap(&arr[neg], &arr[pos]);
        pos++;
        neg += 2;
    }
}
```

Top 30 array questions

https://javarevisited.blogspot.com/2015/06/top-20-array-interview-questions-and-answers.html

String logical questions

Sum of two large numbers stored as string

```
string str1 = "123";
    string str2 = "981";
    string str;
    if(str1.length()>str2.length()){
        string temp=str1;
        str1=str2;
        str2=temp;
    }
    int n1=str1.length();
    int n2=str2.length();
    int diff=n2-n1;
    int carry=0;

    for(int i=n1-1; i>=0; i--){
       int sum=((str1[i]-'0')+(str2[i+diff]-'0')+carry);
        str+=sum%10 + '0';
        carry=sum/10;
    }
    for(int i=n2-n1-1; i>=0; i--){
       int sum=((str2[i]-'0')+carry);
        str+=sum%10 + '0';
        carry=sum/10;
    }
    if(carry) str+=carry+'0';
    int n=str.length();
    for(int i=0;i<n/2;i++){
        char temp=str[i];
        str[i]=str[n-i-1];
        str[n-i-1]=temp;

    }
     cout<<str1<<endl;
     cout<<str2<<endl;
    cout<<"----\n";
    cout<<str<<endl;
    cout<<"----\n";
```

Reverse a string

```
string str="muzammal murtaza";
    int n=str.length();
    for(int i=0;i<n/2;i++){
        char temp=str[i];
        str[i]=str[n-i-1];
        str[n-i-1]=temp;
    }
    for(int i=0;i<n;i++){
        cout<<str[i];
    }
```

Check a string is palindrome or not

```
 string str="aaaa";
    int n=str.length();
    bool flag=true;
    for(int i=0;i<n/2;i++){
        if(str[i]!=str[n-i-1]) {flag=false; break;}
    }
    if(flag) cout<<"palindrome\n";
    else cout<<"no\n";
```

Find first non-repeating character in string

```
#include <iostream>
using namespace std;
int main()
{
    char count[256];
    string str="muzammal";
    for (int i = 0;i<str.length(); i++)
        count[str[i]]++;
      int index = -1, j;
     for (j = 0; j < str.length(); j++) {
        if (count[str[j]] == 1) {
                index = j;
                break;
            }
        }
    cout<<str[index]<<endl;
    return 0;
}
```