**1. Advantages and Disadvantages Of Using Macro and Inline Functions in Embedded?**

**A.** Macros are abbreviations for lengthy and frequently used statements. When a macro is called the entire code is substituted by a single line through the macro definition is of several lines.

The advantage of macro is that it reduces the time taken for control transfer as in the case of function. The disadvantage of it is here the entire code is substituted so the program becomes lengthy if a macro is called several times.

Disadvantage is Macros and inline functions increased the size of executable code. Expressions passed as arguments to inline functions are evaluated only once while expression passed as argument to inline functions are evaluated more than once.

**Inline Function** are those function whose definitions are small and be substituted at the place where its function call is happened. Function substitution is totally compiler choice

Advantages:

Inline function instruct compiler to insert complete body of the function wherever that function got used in code.

1) It does not require function calling overhead.  
2) It also saves overhead of variables push/pop on the stack, while function calling.  
3) It also saves overhead of return call from a function.  
4) It increases locality of reference by utilizing instruction cache.  
5) After in-lining compiler can also apply intraprocedural optimization if specified. This is the most important one, in this way compiler can now focus on dead code elimination, can give more stress on branch prediction, induction variable elimination etc.

**Disadvantages** :-  
1) May increase function size so that it may not fit on the cache, causing lots of cahce miss.  
2) After in-lining function if variables number which are going to use register increases than they may create overhead on register variable resource utilization.  
3) It may cause compilation overhead as if somebody changes code inside inline function than all calling location will also be compiled.  
4) If used in header file, it will make your header file size large and may also make it unreadable.  
5) If somebody used too many inline function resultants in a larger code size than it may cause thrashing in memory. More and more number of page fault bringing down your program performance.  
6) It's not useful for embedded system where large binary size is not preferred at all due to memory size constraints.

Now covering the topic which most the people are interested in the "Performance".  
In most of the cases Inline function boost performance if used cautiously as it saves lots of overhead as discussed in our Advantages section above but as we have also discussed its disadvantages one need to be very cautious while using them. Today's modern compiler inline functions automatically, so no need to specify explicitly in most of the cases. Although placing inline keyword only gives compiler a hint that this function can be optimized by doing in-lining, its ultimately compiler decision to make it inline. Though there are ways to instruct compiler too, for making a function call inline like one can use \_\_forceinline to instruct compiler to inline a function while working with microsoft visual c++. I suggest not to use this keyword until you are very sure about performance gain. Making a function inline may or may not give you performance boost, it all depends on your code flows too. Don't expect a magical performance boost by prepending inline keyword before a function to your code as most of the compiler nowadays does that automatically.  
  
As we have seen inline function serves in terms of performance but one has to use it with extreme cautions.

I have prepared a few guidelines for its use.  
Uses Guidelines :-  
1) Always use inline function when you are sure it will give performance.  
2) Always prefer inline function over macros.  
3) Don't inline function with larger code size, one should always inline small code size function to get performance.  
4) If you want to inline a function in class, then prefer to use inkine keyword outside the class with the function definition.  
5) In c++, by default member function declared and defined within class get lnlined. So, no use to specify for such cases.  
6) Your function will not be inlined in case there is differences between exception handling model. Like if caller function follows c++ structure handling and your inline function follows structured exception handling.  
7) For recursive function most of the compiler would not do in-lining but microsoft visual c++ compiler provides a special pragma for it i.e., pragma inline recursion(on) and once can also control its limit with pragma inline depth.  
8) If the function is virtual and its called virtually then it would not be in lined. So, take care for such cases, same hold true for the use of function pointers.

**2. Why Cannot Arrays Be Passed by Values to Functions with example?**

**Answer**. Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

## Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [ arraySize ];

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement −

double balance[10];

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

## Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

balance[4] = 50.0;

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above −



## Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

double salary = balance[9];

The above statement will take the 10th element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays

Arrays can't be passed by values. Because, the array name is evaluated to be a pointer to the first element of the array. ... Its type is, therefore, int \*, and a called function uses this pointer (passed as an argument) to indirectly access the elements of the array.

In C++ or C, if your array is being passed in a function, the pointer to the array gets passed as value, so the called function manipulations would directly refer to the array declared in calling function. The reasons are:

An array is basically contiguous memory locations accessed by a base address, so whenever an array is passed in a function, the array decays into pointer.

It is not possible to determine the length of the array for creating an exact copy of the array and it is also not advisable to have such things because of bad memory management.

However, in C++ now we have various classes like vector, array which can be passed by value as they have a valid constructor and can be safely copied as a value to a different memory location.

Example:

#include <stdio.h>

// Note that arr[] for fun is just a pointer even if square

// brackets are used

void fun (int arr[]) // SAME AS void fun (int \*arr)

{

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

printf("\array size inside fun () is %d", n);

}

// Driver program

int main ()

{

int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};

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

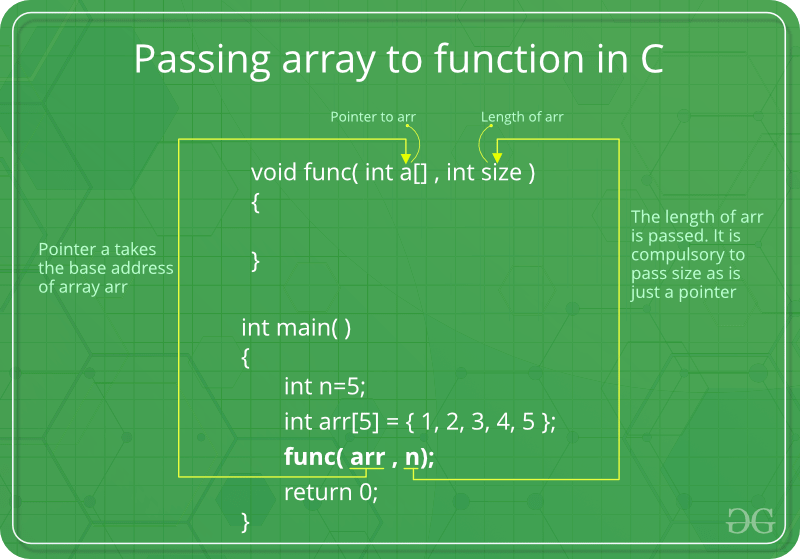
printf("Array size inside main () is %d", n);

fun(arr);

return 0;

}

Therefore, in C, we must pass size of array as a parameter. Size may not be needed only in case of ‘\0’ terminated character arrays, size can be determined by checking end of string character.



Arrays are not passed by value because arrays are essentially continuous blocks of memory. If you had an array you wanted to pass by value, you could declare it within a structure and then access it through the structure