Properties of Arrays in C++

- An Array is a collection of data of the same data type, stored at a contiguous memory location.
- Indexing of an array starts from 0. It means the first element is stored at the 0th index, the second at 1st, and so on.
- Elements of an array can be accessed using their indices.
- Once an array is declared its size remains constant throughout the program.
- An array can have multiple dimensions.
- The number of elements in an array can be determined using the sizeof operator.
- We can find the size of the type of elements stored in an array by subtracting adjacent addresses.
- There are 3 types of arrays
    - Single Dimensional Array
    - Two Dimensional Array
    - Multi Dimensional Array
- In C++, we can declare an array by simply specifying the data type first and then the name of an array with its size.

  data_type array_name[size_of_array]

  int arr[5];

Initialization of Array in C++

- In C++, we can initialize an array in many ways but we will discuss some most common ways to initialize an array. We can initialize an array at the time of declaration or after declaration.
    - int arr[5] = {1, 2, 3, 4, 5};
    - int arr[ ] = {1, 2, 3, 4, 5};
    - int arr[N];

  ```
  for(int i =0; i<N; i++)  {
          arr[i] = value;
  }
  ```

- Initialize array partially
    - int partialArr[5] = {1, 2};
- Initialize array to 0

    - int zeroArr[5] = {0}; //all elements will be 0, this will happen only for 0
- Acessing an element of an Array
    - Elements of an array can be accessed by specifying the name of the array, then the index of the element enclosed in the array subscript operator []. For example, arr[i].

Relation between Arrays and Pointers in C++

- In C++, arrays and pointers are closely related to each other. The array name is treated as a pointer that stored the memory address of the first element of the array.
- As we have discussed earlier, In array, elements are stored at contiguous memory locations that's why we can access all the elements of an array using the array name.
- cout << "first element: " << *arr << endl;
- cout << "Second element: " << *(arr + 1) << endl;
- cout << "Third element: " << *(arr + 2) << endl;
- cout << "fourth element: " << *(arr + 3) << endl;
- In the above code, we first declared an array "arr" with four elements. After that, we are printing the array elements.
- Array name is a pointer that stores the address of the first element of an array so, to print the first element we have dereferenced that pointer (*arr) using dereferencing operator (*) which prints the data stored at that address.
- To print the second element of an array we first add 1 to arr which is equivalent to (address of arr + size_of_one_element *1) that takes the pointer to the address just after the first one and after that, we dereference that pointer to print the second element. Similarly, we print rest of the elements of an array without using indexing.

Multidimensional Arrays in C++

- Arrays declared with more than one dimension are called multidimensional arrays.
- The most widely used multidimensional arrays are 2D arrays and 3D arrays.
- These arrays are generally represented in the form of rows and columns.
    - Data_type Array_name[Size1][Size2]….[SizeN];

Two Dimensional Array in C++

- A two-dimensional array is a grouping of elements arranged in rows and columns.
- The left index indicates the row, and right index indicates column
- Starting index of matrix or array is always 0.
- Each element is accessed using two indices: one for the row and one for the column, which makes it easy to visualize as a table or grid.

  data_type array_name[n][m];

  where n – is number of rows

m – is number of columns

Functions in C++

- A function is a set of statements that takes input, does some specific computation, and produces output.
- The idea is to put some commonly or repeatedly done tasks together to make a function, so that instead of writing the same code again and again for different inputs, we can call this function.
- In simple terms, a function is a block of code that runs only when it is called.

  data_type func_name(data_type var1, data_type var2);

Why Do We Need Functions?

- Functions help us in *reducing code redundancy*. If functionality is performed at multiple places in software, then rather than writing the same code, again and again, we create a function and call it everywhere. This also helps in maintenance as we have to make changes in only one place if we make changes to the functionality in future.
- Functions make code *modular*. Consider a big file having many lines of code. It becomes really simple to read and use the code, if the code is divided into functions.
- Functions provide *abstraction*. For example, we can use library functions without worrying about their internal work.

Function Declaration

- A function declaration tells the compiler about the number of parameters, data types of parameters, and returns type of function. Writing parameter names in the function declaration is optional but it is necessary to put them in the definition.

```
// C++ Program to show function that takes
// two integers as parameters and returns
// an integer
int max(int, int);


// A function that takes an int
// pointer and an int variable
// as parameters and returns
// a pointer of type int
int* swap(int*, int);


// A function that takes
// a char as parameter and
```

```
// returns a reference variable
char* call(char b);


// A function that takes a
// char and an int as parameters
// and returns an integer
int fun(char, int);
```

User Defined Function

- User-defined functions are user/customer-defined blocks of code specially customized to reduce the complexity of big programs.
- They are also commonly known as "*tailor-made functions*" which are built only to satisfy the condition in which the user is facing issues meanwhile reducing the complexity of the whole program.

Library Function

- Library functions are also called "*built-in Functions*".
- These functions are part of a compiler package that is already defined and consists of a special function with special and different meanings.
- Built-in Function gives us an edge as we can directly use them without defining them whereas in the user-defined function we have to declare and define a function before using them.
  For Example: sqrt(), setw(), strcat(), etc.

Parameter Passing to Functions

- The parameters passed to the function are called *actual parameters*.
- The parameters received by the function are called *formal parameters*.

Passing Parameters

- There are two most popular ways to pass parameters:

1. *Pass by Value:* In this parameter passing method, values of actual parameters are copied to the function's formal parameters. The actual and formal parameters are stored in different memory locations so any changes made in the functions are not reflected in the actual parameters of the caller.

2. *Pass by Reference:* Both actual and formal parameters refer to the same locations, so any changes made inside the function are reflected in the actual parameters of the caller.

Pass by value

```
// C++ Program to demonstrate function definition
#include <iostream>
```

```
using namespace std;

void fun(int x)
{
    // definition of function
    x = 30;
}

int main()
{
    int x = 20;
    fun(x);
    cout << "x = " << x;   //the value of x is not modified using the function fun().
    return 0;
}
```

Functions Using Pointers

- The function fun() expects a pointer ptr to an integer (or an address of an integer).
- It modifies the value at the address ptr.
- The dereference operator * is used to access the value at an address.
- In the statement '*ptr = 30', the value at address ptr is changed to 30.
- The address operator & is used to get the address of a variable of any data type.
- In the function call statement 'fun(&x)', the address of x is passed so that x can be modified using its address.

```
// C++ Program to demonstrate working of
// function using pointers
#include <iostream>
using namespace std;

void fun(int* ptr) { *ptr = 30; }

int main()
{
```

```
    int x = 20;
    fun(&x);
    cout << "x = " << x;

    return 0;
}
```

Difference between call by value and call by reference

Call by Value
A copy of the value is passed to the function
Changes made inside the function are not reflected on other functions
Actual and formal arguments will be created at different memory location

Call by Reference
An address of value is passed to the function
Changes made inside the function are reflected outside the function as well
Actual and formal arguments will be created at same memory location.

Points to Remember About Functions

- Most C++ program has a function called main() that is called by the operating system when a user runs the program.
- Every function has a return type. If a function doesn't return any value, then void is used as a return type. Moreover, if the return type of the function is void, we still can use the return statement in the body of the function definition by not specifying any constant, variable, etc. with it, by only mentioning the 'return;' statement which would symbolize the termination of the function
- To declare a function that can only be called without any parameter, we should use "void fun(void)". As a side note, in C++, an empty list means a function can only be called without any parameter. In C++, both void fun() and void fun(void) are same.

Main Function

- The main function is a special function. Every C++ program must contain a function named main. It serves as the entry point for the program. The computer will start running the code from the beginning of the main function.

Types of Main Functions

- Without parameters:
  - int main() { …. Return 0;}
- With parameters:
  - int main(int argc, char* const argv[]) { …. Return 0;}

- argc – Non negative value representing the number of arguments passed to the program from the environment in which the program is run
- argv – pointers to the first element of an array
- The reason for having the parameter option for the main function is to allow input from the command line.
- When you use the main function with parameters, it saves every group of characters (separated by a space) after the program name as elements in an array named argv.

C++ Passing Array to Function

- In C++, to reuse the array logic, we can create a function. To pass an array to a function in C++, we need to provide only the array name.

  function_name(array_name[]); //passing array to a function

C++ Overloading (Function)

- If we create two or more members having the same name but different in number or type of parameters, it is known as C++ overloading. In C++, we can overload:
  - *methods,*
  - *constructors and*
  - *indexed properties*
- Types of overloading in C++ are:
  - *Function overloading*
  - *Operator overloading*

C++ Function Overloading

- Function Overloading is defined as the process of having two or more functions with the same name, but different parameters.

- In function overloading, the function is redefined by using either different types or number of arguments.
- It is only through these differences a compiler can differentiate between the functions.
- The advantage of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action.

Function Overloading and Ambiguity

- When the compiler is unable to decide which function is to be invoked among the overloaded function, this situation is known as *function overloading ambiguity.*
- When the compiler shows the ambiguity error, the compiler does not run the program

Function with Default Arguments

```
#include <iostream>
using namespace std;
void fun(int);
void fun(int, int);
void fun(int i) { cout << "Value of i is : " << i << endl; }
void fun(int a, int b = 9)
{
   cout << "Value of a is : " << a << endl;
   cout << "Value of b is : " << b << endl;
}
int main()
{
   fun(12);

   return 0;
}
```

- shows an error "*call of overloaded 'fun(int)' is ambiguous*".
- The fun(int i) function is invoked with one argument.
- The fun(int a, int b=9) can be called in two ways:
    - first is by calling the function with one argument, i.e., fun(12) and
    - another way is calling the function with two arguments, i.e., fun(4,5).
- Therefore, the compiler could not be able to select among fun(int i) and fun(int a,int b=9).

Function with Pass By Reference

```
#include <iostream>
using namespace std;
void fun(int);
void fun(int&);
int main()
{
   int a = 10;
   fun(a); // error, which fun()?
   return 0;
}
void fun(int x) { cout << "Value of x is : " << x << endl; }
void fun(int& b)
{
   cout << "Value of b is : " << b << endl;
}
```

error "*call of overloaded 'fun(int&)' is ambiguous*"

- The first function takes one integer argument and the second function takes a reference parameter as an argument.
- In this case, the compiler does not know which function is needed by the user as there is no syntactical difference between the fun(int) and fun(int &).

Inline Functions in C++

- C++ provides inline functions to reduce the function call overhead.
- An inline function is a function that is expanded in line when it is called.
- When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call.
- This substitution is performed by the C++ compiler at compile time.
- An inline function may increase efficiency if it is small.

```
inline return_type function_name(parameters)
        {…….}
```

- Remember, inlining is only a request to the compiler, not a command. The compiler can ignore the request for inlining.

- The compiler may not perform inlining in such circumstances as
    - If a function contains a loop. (*for, while and do-while*)
    - If a function contains static variables.
    - If a function is recursive.
    - If a function return type is other than void, and the return statement doesn't exist in a function body.
    - If a function contains a switch or goto statement.

Why Inline Functions are Used?

- For functions that are large and/or perform complex tasks, the overhead of the function call is usually insignificant compared to the amount of time the function takes to run. However, for small, commonly-used functions, the time needed to make the function call is often a lot more than the time needed to actually execute the function's code. This overhead occurs for small functions because the execution time of a small function is less than the switching time.

Inline functions Advantages

- Function call overhead doesn't occur.
- It also saves the overhead of push/pop variables on the stack when a function is called.
- It also saves the overhead of a return call from a function.
- When you inline a function, you may enable the compiler to perform context-specific optimization on the body of the function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of the calling context and the called context.
- An inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function called preamble and return.

Inline function Disadvantages

1. The added variables from the inlined function consume additional registers, After the in-lining function if the variable number which is going to use the register increases then they may create overhead on register variable resource utilization. This means that when the inline function body is substituted at the point of the function call, the total number of variables used by the function also gets inserted. So the number of registers going to be used for the variables will also get increased. So if after function inlining variable numbers increase drastically then it would surely cause overhead on register utilization.
2. If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of the same code.
3. Too much inlining can also reduce your instruction cache hit rate, thus reducing the speed of instruction fetch from that of cache memory to that of primary memory.
4. The inline function may increase compile time overhead if someone changes the code inside the inline function then all the calling location has to be recompiled

because the compiler would be required to replace all the code once again to reflect the changes, otherwise it will continue with old functionality.

5. Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.
6. Inline functions might cause thrashing because inlining might increase the size of the binary executable file. Thrashing in memory causes the performance of the computer to degrade. The following program demonstrates the use of the inline function.

Default Arguments in C++

- A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn't provide a value for the argument. In case any value is passed, the default value is overridden.

eg.

int sum(int x, int y, int z = 0, int w = 0)