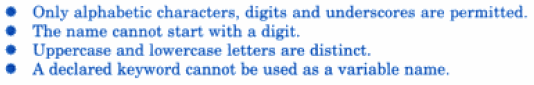


**TOKENS:** The smallest individual units in a program are known as tokens

**KEYWORDS:** They are explicitly reserved identifiers and cannot be used as names for the program variables or other user-defined program elements

**IDENTIFIERS:** It refers to the names of variables, functions, arrays, classes created by programmer

**Rules of Identifiers:**



**ARRAYS:**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the number of elements it should store: string cars[4];

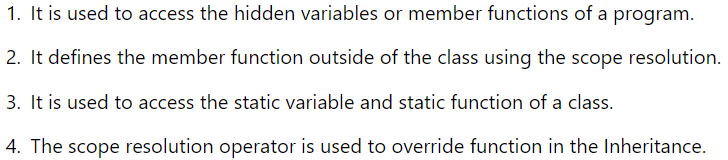
**FUNCTIONS:**

A function is a block of code that performs some operation. A function can optionally define input parameters that enable callers to pass arguments into the function. A function can optionally return a value as output.

**Scope Resolution Operator**

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

**Uses of the scope resolution Operator**



**Why does C++ have type modifiers?**

C++ allows the char, int, and double data types to have modifiers preceding them. Type Modifiers are special keywords defined in the programming language which are used to modify the default properties of the Built-in Data types. Type Modifiers are special keywords that are used to modify the range of the data types and also the memory space allocated to the variable.

**Why is an array called a derived data type?**

An array is a derived data type because it cannot be defined on its own, it is a collection of basic data types usually, such as integers, doubles, floats, booleans, etc.

**The size of a char array that is declared to store a string should be one larger than the number of characters in the string. Why?**

String is represented as a character array, which is terminated by a null character '\0'. You do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Due to this this null character, array should be one larger than the number of characters in the string.

**How does a constant defined by const differ from the constant defined by the preprocessor statement #define?**

CONSTs are handled by the compiler, whereas #DEFINEs are handled by the pre-processor. The big advantage of const over #define is type checking. #defines can’t be type checked, so this can cause problems when trying to determine the data type. If the variable is, instead, a constant then we can grab the type of the data that is stored in that constant variable.

**What do you mean by dynamic initialization of a variable? Give an example**

The process of initializing a variable now it is declared at runtime is called dynamic initialization of the variable. Thus, during the dynamic initialization of a variable, a value is assigned to execution when it is declared. Dynamic Initialization is also known as Implicit Initialization.

Int a;

Cout<< “Enter Value of a”;

Cin>>a;

Int cube = a\*a\*a;

**What do you mean by static initialization of a variable? Give an example**

In this method, the variable is assigned a value in advance. Here, the values are assigned in the declaration statement. Static Initialization is also known as Explicit Initialization.

Int a;

a=5;

int b = 10;

int x =4, y = 5;

**What is a reference variable? What is its major use?**

Reference variable is an alternate name of already existing variable. It cannot be changed to refer another variable and should be initialized at the time of declaration and cannot be NULL. The operator '&' is used to declare reference variable. The following is the syntax of reference variable. A reference variable is one that refers to the address of another variable. It represents the name of another variable, location, or value. Once you initialize the variable references, that variable will be referred to using the variable name or reference name.

**List at least 4 new operators added by C++ which aid oop?**

The new operators in C++ (but not in C) are new, delete, compl, and, and\_eq, not, not\_eq, or, or\_eq, xor, xor\_eq, bitand and bitor. Of those only the first two can really be said to aid OOP. However, other keywords that specifically aid OOP include class, friend, mutable, private, protected, public and template.

**What is the application of the scope resolution operator in C++?**

Scope resolution operator in C++ can be used for: Accessing a global variable when there is a local variable with same name. Defining a function outside a class. Accessing a class's static variables. The operator is represented as the double colon (::) symbol.

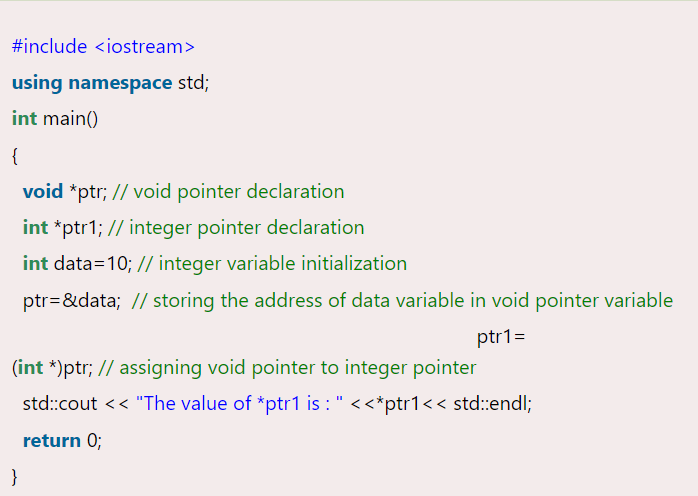
**Applications of void data type:**

* Function Return Type: You can use void as the return type for functions that do not return a value.
* Pointer to Void: A pointer of type void\* can point to objects of any data type. This is often used for memory allocation and dynamic memory management
* Dynamic Memory Allocation: The void\* pointer is often used with memory allocation functions like malloc and calloc
* Function Pointers: Function pointers can be of type void\* to create more flexible function dispatch mechanisms
* Callback Mechanisms: When implementing callback mechanisms or event handlers, you can use void\* to pass user-defined data to callback functions

**Can we assign a void pointer to an int type pointer? If not, why? How can we achieve this?**

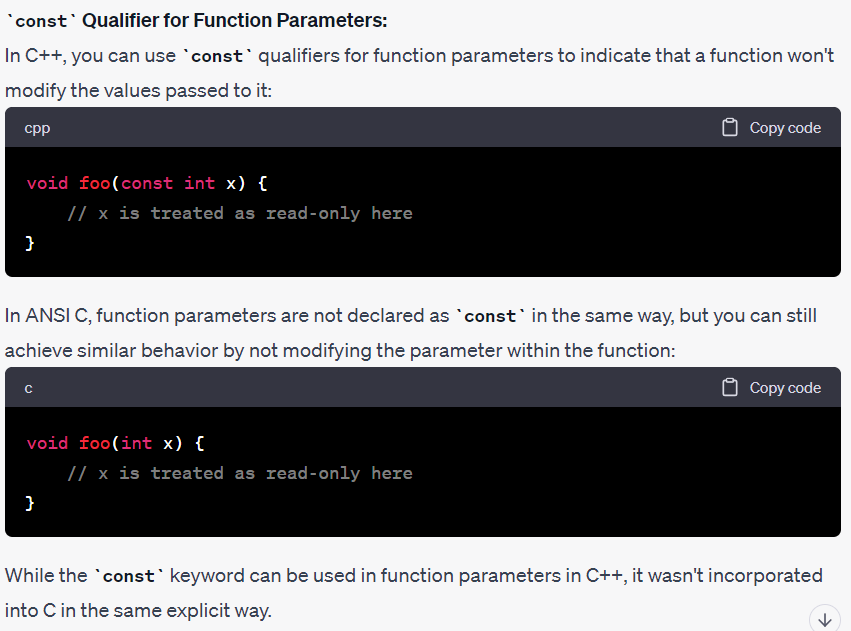
No, you cannot directly assign a void\* pointer to an int\* pointer in C++. This is because void\* is a generic pointer type that doesn't have type information, while int\* is a pointer specifically designed to point to integers. Attempting to directly assign a void\* pointer to an int\* pointer would result in a compilation error because the types are not compatible.

To achieve this, you need to perform a type conversion or cast explicitly. You should only do this if you are absolutely certain that the memory location pointed to by the void\* pointer indeed contains an integer. Here's how you can achieve it:



**The const was taken from c++ and incorporated in ANSI C, although quite differently. Explain**

The most important difference is that in C++ a const variable is a constant expression (even prior the introduction of C++11 constexpr), but a const variable in C is not. Meaning that C++ allows you to do things like const size\_t n = 1; static int array[n]; but C does not allow that.

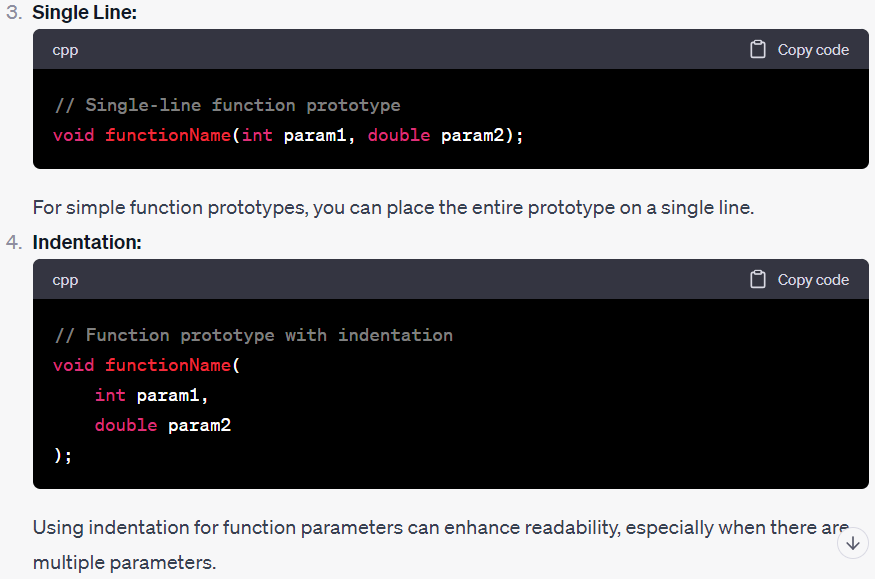
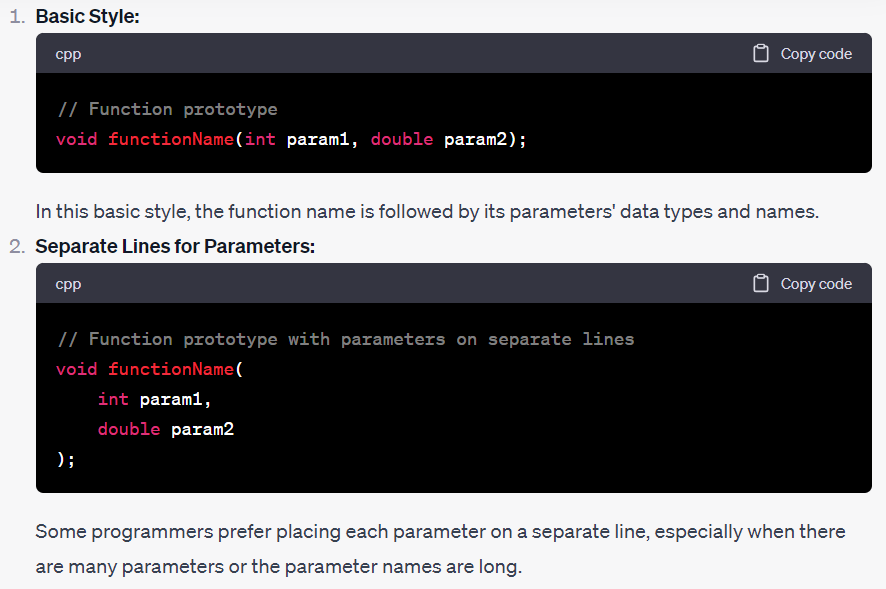


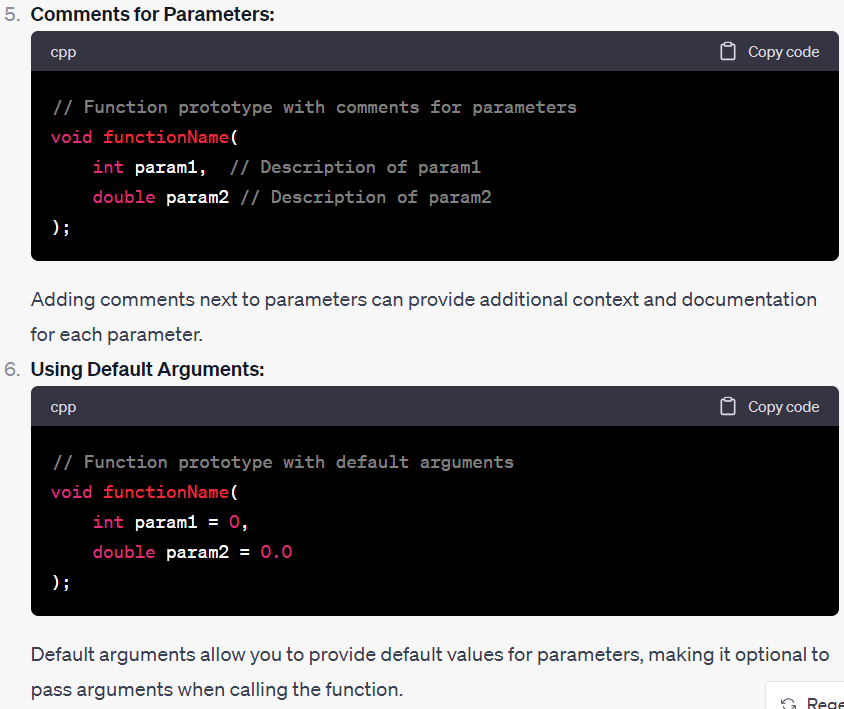
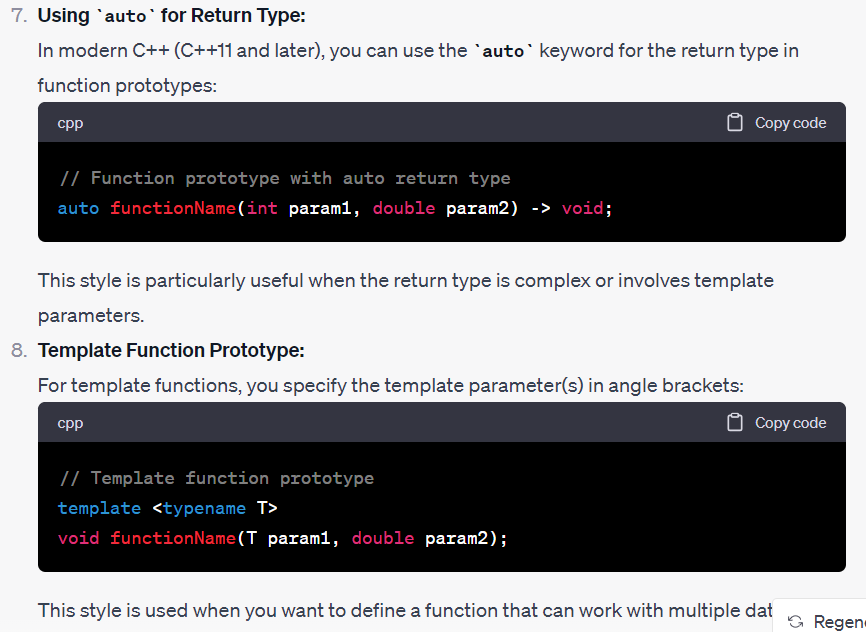
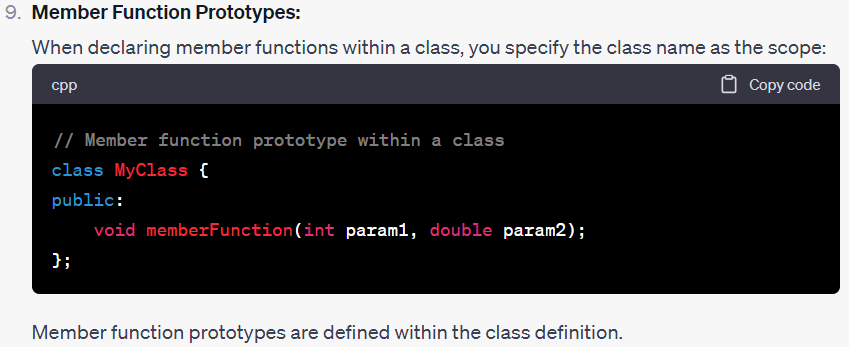
**In C++, a variable can be declared anywhere in the scope. What is the significance of this feature?**

* Limited Scope: Variables declared within a smaller scope have limited visibility. They are accessible only within the block where they are defined and any nested blocks.
* Resource Management: Variables with limited scope are automatically destroyed when they go out of scope. This feature is especially useful for resource management, like releasing memory allocated on the heap when a function exits or closing files when they are no longer needed.
* Reduces Name Conflicts: By allowing variable declarations within blocks, you can reuse variable names in different blocks without causing naming conflicts.
* Control Flow and Conditional Initialization: Variables declared within blocks can be conditionally initialized based on program logic.
* Clean Code and Readability: Local scope variable declarations can lead to cleaner and more readable code.
* Optimization: Modern C++ compilers are often able to optimize variables with limited scope more effectively.
* Reduced Lifetime: Variables with limited scope have shorter lifetimes, which can help prevent bugs related to using variables after they are no longer needed.

**What are the advantages of function prototypes in C++? (REMEMBER ANY 5)**

* Early Detection of Errors: Function prototypes provide a clear and explicit declaration of a function's interface before it's used in the program. This helps the compiler catch errors like incorrect argument types, mismatched return types, or missing functions during compilation, rather than at runtime
* Code Organization: Prototypes help organize your code by separating the declaration of functions from their actual implementations. This can make your code easier to navigate, especially in large projects with many functions.
* Readability: Prototypes improve code readability by providing a summary of a function's purpose and signature. This allows programmers to quickly understand how to use a function without needing to look up its implementation.
* Documentation: Prototypes can serve as a form of documentation. By reading the prototype, you can see the function's name, return type, and parameter list, which helps clarify the function's intended behavior.
* Encapsulation: Prototypes can be placed in header files, making them accessible to other parts of the program. This encapsulation allows you to hide the implementation details of a function while providing a public interface for other parts of the code to use.
* Facilitates Function Overloading: In C++, you can define multiple functions with the same name but different parameter lists (function overloading). Function prototypes are essential for specifying the different parameter lists and enabling the compiler to determine which function to call based on the arguments passed.
* Cross-File Compilation: When working with multiple source files, function prototypes in header files allow you to call functions defined in other files. This supports modular code development and helps prevent issues related to calling undefined functions.
* Forward Declarations: In some cases, you may want to use a function before its full implementation is available, especially when dealing with mutually recursive functions or circular dependencies. Function prototypes provide a way to forward declare functions, allowing you to use them before their actual definitions.
* Type Safety: Prototypes ensure type safety by specifying the correct data types for function arguments and return values. This helps catch type-related errors early in the development process.
* Tool Support: Many development tools, including integrated development environments (IDEs) and code editors, rely on function prototypes to provide auto-completion, code suggestions, and documentation tooltips, making coding more efficient.

**Describe the different styles of writing prototypes? (Remember any 5)** 

**What is the main advantage of passing arguments by reference?**

* Efficiency: When you pass an argument by reference, you avoid the overhead of copying large or complex data structures, such as arrays, objects, or other user-defined types.
* Avoids Data Duplication: By passing by reference, you ensure that there is only one copy of the data in memory. This is important when working with resource-intensive data structures or limited memory environments.
* Modifiability: When you pass by reference, you can modify the original data directly within the function, and those modifications are reflected outside the function.
* Saves Memory: Passing large objects or arrays by reference can save a significant amount of memory. This is especially relevant when dealing with data that cannot or should not be duplicated in memory.
* Consistency: It ensures consistency in your data because you are working with the same instance of the data both inside and outside the function.
* Avoids Object Slicing: When you pass derived class objects by reference to functions that accept base class references or pointers, you prevent object slicing.
* Enables Function Overloading: Pass-by-reference is essential for function overloading with different parameter types. Without it, function overloading might be limited to overloading by value or pointer, which can be less intuitive.
* Readability: Pass-by-reference can make function calls more readable because they clearly indicate that the function may modify the original data.

**When will you make a function inline? Why? (Remember any 5)**

* Small Functions: Inline functions are most effective for small functions, especially those with a minimal amount of code. Inlining larger functions can lead to code bloat and reduced performance gains.
* Performance Optimization: The primary reason for using inline functions is to improve performance. By marking a function as inline, you suggest to the compiler that it should replace function calls with the actual code of the function at the call site.
* Frequent Function Calls: Functions that are called frequently in a program, particularly in tight loops, benefit the most from being declared as inline. This is because the overhead of function call and return becomes noticeable in such cases.
* Access to Enclosing Scope: Inline functions have access to the enclosing scope, which means they can directly access variables and data in the calling function, potentially eliminating the need for parameter passing and providing performance benefits.
* Compiler's Discretion: It's important to note that marking a function as inline is a suggestion to the compiler, not a command. The compiler may choose to inline the function or not, depending on its optimization settings and the complexity of the function.
* Header-Only Libraries: In some cases, inline functions are used in header-only libraries, where the entire implementation is included in header files.
* Templates: Functions defined within template classes are often implicitly inline because their definitions are typically provided in header files.
* Avoiding Multiple Definitions: When you define a function in a header file that is included in multiple source files, declaring it as inline helps prevent multiple definitions of the same function when linking the object files.

**How does an inline function differ from a preprocessor macro?**

The main difference between inline and macro functions is that inline functions are parsed by the compiler, whereas macros in a program are expanded by the preprocessor. The keyword "inline" is used to define an inline function, whereas "#define" is used to define a macro.

**When should I use macro functions?**

Macros are used for short operations and to avoid the overhead of function calls. It can be used if a short operation is repeated in a program. When the same block of code must be executed multiple times, function-like macros can be helpful.

**When should I use inline functions over macros?**

Use Inline Functions when you need the benefits of function call optimization without sacrificing type safety and debugging capabilities. Inline Functions are suitable for more complex logic and situations where code readability and maintainability are important considerations.

**When do we need to use default arguments in a function? (Remember any 5)**

* Optional Parameters: Default arguments enable you to make function parameters optional. Users of the function can choose to provide values for some parameters while omitting others, simplifying function calls.
* Overloading Reduction: Instead of defining multiple overloaded functions with different parameter combinations, you can use default arguments to consolidate similar functionality into a single function.
* Backward Compatibility: When modifying existing code, you can add default arguments to functions without breaking existing code that calls those functions without providing the new parameters. This maintains backward compatibility.
* Simplifying Interfaces: Default arguments can simplify the public interface of a function or class, reducing the number of overloaded methods and making the API more intuitive.
* Reducing Code Duplication: Default arguments can help reduce code duplication in functions with similar logic but slight variations in parameter handling.
* Complex Constructors: Default arguments can be used in constructors to provide reasonable default values for class members. This simplifies object creation and initialization.
* Configurable Behavior: Default arguments can be used to specify configurable behavior or settings in functions. Users can choose to accept the defaults or customize the behavior by providing different values.
* Minimizing Function Overloads: In some cases, default arguments can help reduce the number of function overloads in your codebase, making the codebase more manageable.

**What is the significance of an empty parenthesis in a function declaration?**

* Clear Indication: It provides a clear and explicit indication that the function does not expect any parameters to be passed when it is called. This can be helpful for both the programmer who defines the function and the programmer who uses it, as it establishes a clear contract regarding the function's behavior.
* Simplified Function Call: When calling a function with an empty parameter list, you can simply use the function name followed by empty parentheses, without the need to specify any arguments.
* Code Clarity: It improves code clarity and readability by making it evident that the function's behavior is independent of any external input. When reading or maintaining code, seeing an empty parameter list immediately conveys that no external data is needed for the function to execute.
* Consistency: Using empty parentheses for functions without parameters aligns with the conventions of the C++ language and is consistent with the way the standard library functions are often declared.

**What do you meant by overloading of a function? When do we use this concept?**

Function overloading is a concept in programming where multiple functions in the same scope or class have the same name but differ in the number or type of their parameters. This allows you to define multiple functions with the same name, each tailored to handle different argument types or numbers of arguments.

* Improved Code Readability: Overloading can make your code more intuitive and readable. You can have a single function name for operations that conceptually do the same thing but with different types of data. For example, you can have an add function that works for both integers and floating-point numbers.
* Default Arguments: Overloading can be used to provide default arguments to a function. For instance, you can have an overloaded function with a version that takes one argument and another version that takes two arguments with one of them being optional.
* Consistent Naming: It allows you to maintain consistent naming conventions for functions that perform similar tasks. Instead of having to create distinct names for similar functions, you can reuse a meaningful function name.

**How do structures in C and C++ differ?**

