Sure, I can help you with that. Here is a table that maps the different elements of a C program to the three different ways they can be built:

| Element | Shared Library (Loaded Explicitly) |

| | Shared Library (Loaded Implicitly) |

| | Static Library |

|-----------------------|-------------------------------------|

| Static function | Defined in the shared library |

| | Defined in the shared library |

| | Defined in the static library |

| Extern function | Defined in the main program |

| | Defined in the main program |

| | Defined in the static library |

| Heap variable | Allocated in the heap of the main |

| | program |

| | Allocated in the heap of the main |

| | program |

| | Not allocated |

| Const local variable | Defined in the stack of the main |

| | program |

| | Defined in the stack of the main |

| | program |

| | Defined in the static data segment |

| | of the static library |

| Non-const local | Defined in the stack of the main |

| variable | program |

| | Defined in the stack of the main |

| | program |

| | Not allocated |

| Const global variable | Defined in the static data segment |

| | of the shared library |

| | Defined in the static data segment |

| | of the shared library |

| | Defined in the static data segment |

| | of the static library |

| Non-const global | Defined in the static data segment |

| variable | of the shared library |

| | Defined in the static data segment |

| | of the shared library |

| | Defined in the static data segment |

| | of the static library |

| Static local variable | Defined in the static data segment |

| | of the main program |

| | Not allocated |

| | Not allocated |

| Static global | Defined in the static data segment |

| variable | of the main program |

| | Defined in the static data segment |

| | of the main program |

| | Defined in the static data segment |

| | of the static library |

| String literals | Located in the static data segment |

| | of the shared library |

| | Located in the static data segment |

| | of the shared library |

| | Located in the static data segment |

| | of the static library |

| Environment variables | N/A |

| | N/A |

| | N/A |

| Command line arguments| N/A |

| | N/A |

| | N/A |

Here is a brief explanation of each of these mappings:

\* \*\*Static function:\*\* A static function is a function that is defined in the current file and is not visible to other files. When a shared library is loaded explicitly, all of the static functions in the library are made visible to the main program. When a shared library is loaded implicitly, only the functions that are explicitly referenced by the main program are made visible. Static functions are always allocated in the static data segment of the library or program in which they are defined.

\* \*\*Extern function:\*\* An extern function is a function that is defined in another file and is visible to the current file. When a shared library is loaded explicitly, all of the extern functions in the library are made visible to the main program. When a shared library is loaded implicitly, only the extern functions that are explicitly referenced by the main program are made visible. Extern functions are always allocated in the static data segment of the library in which they are defined.

\* \*\*Heap variable:\*\* A heap variable is a variable that is allocated in the heap, which is a region of memory that is managed dynamically by the operating system. Heap variables are not allocated in any of the libraries or the main program.

\* \*\*Const local variable:\*\* A const local variable is a local variable that is declared with the `const` keyword. Const local variables are stored in the stack of the main program and are not allocated in any of the libraries.

\* \*\*Non-const local variable:\*\* A non-const local variable is a local variable that is not declared with the `const` keyword. Non-const local variables are stored in the stack of the main program and are not allocated in any of the libraries.

\* \*\*Const global variable:\*\* A const global variable is a global variable that is declared with the `const` keyword. Const global variables are stored in the static data segment of the library or program in which they are defined.

\* \*\*Non-const global variable:\*\* A non-const global variable is a global variable that is not declared with the `const` keyword. Non-const global variables are stored in the static data segment of the library or program in which they are defined.

\* \*\*Static local variable:\*\* A static local variable is a local variable that is declared with the `static` keyword. Static local variables are stored in the static data segment of the main program.

\* \*\*Static global variable:\*\* A static global variable is a global variable that is declared with the `static` keyword. Static global variables are stored in the static data segment of the library or program in which they are defined.

"String literals," "Environment variables," and "Command line arguments" are typically located:

:###########################String Literals:###########################

Shared Library (Loaded Explicitly): String literals are usually located in the static data segment of the shared library. These literals are part of the library's read-only data.

Shared Library (Loaded Implicitly): Just like when loaded explicitly, string literals in a shared library loaded implicitly are typically located in the static data segment of the shared library.

Static Library: String literals in a static library are also typically located in the static data segment of the static library. When you link with a static library, the string literals become part of the resulting executable.

:###########################Environment variables:###########################

Environment variables are not usually associated with shared libraries. They are typically set and managed at the operating system level and are accessible to all programs.

Environment variables are typically stored in the process's environment block, which is also part of the process's memory.

The environment block is an array of null-terminated strings, where each string represents an environment variable in the format "name=value."

When a process starts, the operating system provides it with a copy of the environment block, which includes variables like PATH, HOME, and others.

Environment variables can be accessed using system calls or functions provided by the operating system, such as getenv in C.

:###########################Command line arguments:###########################

Command line arguments are not related to the loading of shared libraries. They are parameters passed to the main program when it is executed.

Command line arguments are typically stored in an array of strings in memory. Each string in the array corresponds to one argument provided to the program.

The operating system parses the command line when launching a program and creates this array, populating it with the values provided at runtime.

In C and C++ programs, you can access command line arguments through the argc (argument count) and argv (argument vector) parameters of the main function. argc represents the number of arguments, and argv is an array of strings containing the actual argument values.

Command line arguments are typically stored in the stack memory of the process. When a program is executed, the operating system sets up the program's initial stack, which includes space for command line arguments.

The main function in C and C++ programs receives a pointer to the first argument (usually named argv) and an integer representing the number of arguments (usually named argc).

The argv array is stored in the stack, with each element pointing to a separate string containing an argument.

The strings themselves (the actual argument values) are also stored in the stack, and argv holds pointers to these strings.