Project Plan

Domain: Description of the domain and motivation of the need for a General Purpose Language.

A general-purpose language is a programming language that can be used for a wide range of applications, as opposed to specialized languages designed for specific tasks or industries. General-purpose languages are designed to be versatile and flexible, allowing programmers to use them for a variety of purposes, including web development, mobile app development, scientific computing, and many others.

The need for a general-purpose language arises from the fact that different programming languages are optimized for different tasks. For example, a language like SQL is optimized for working with databases, while a language like MATLAB is optimized for scientific computing. While these languages are very powerful for their specific domains, they may not be suitable for other tasks.
A general-purpose language, on the other hand, can be used for a wide range of tasks, making it a more versatile option for programmers. This is particularly useful for smaller projects or projects that don't require specialized features, as it can be more efficient to write the code in a general-purpose language than to learn and use a specialized language.

Additionally, general-purpose languages are often easier to learn and more widely used, which means that there are more resources available for programmers who need to learn or use them. This can be particularly beneficial for beginners or for teams that need to collaborate on projects using a common language.
Overall, the domain of general-purpose languages is vast and covers a wide range of programming tasks. The motivation for the need of a general-purpose language is to provide programmers with a flexible and versatile tool that can be used for a variety of tasks, making programming more efficient, accessible, and widely applicable.

How could domain-experts use and benefit from a GPL?

Domain experts can benefit greatly from using a general-purpose language in a number of ways:
Increased Flexibility: Domain experts often have unique requirements that may not be met by specialized languages. A general-purpose language can provide more flexibility and allow domain experts to create custom solutions that meet their specific needs.

Improved Communication: Domain experts may need to communicate their ideas and requirements to developers or other stakeholders who may not be familiar with specialized languages. Using a general-purpose language can improve communication and make it easier for non-experts to understand the code.

Improved Efficiency: A general-purpose language can make it easier for domain experts to work with data and perform analysis. For example, a data scientist might use a general-purpose language like Python to automate data cleaning and analysis tasks, which can save time and reduce errors.

Access to a Wider Range of Tools and Libraries: General-purpose languages like Python and Java have large ecosystems of tools and libraries that can be used for a wide range of tasks. This can make it easier for domain experts to leverage existing solutions and build custom applications more quickly and efficiently.

Greater Control and Customizability: In some cases, domain experts may need more control over the underlying code or data structures than is possible with specialized languages. A general-purpose language can provide greater control and allow for more customizability, which can be important for specialized applications.

Overall, domain experts can benefit from using a general-purpose language by gaining greater flexibility, improving communication, increasing efficiency, accessing a wider range of tools and libraries, and gaining greater control over the underlying code and data structures.

Language overview:
What is the basic computation that the GPL performs (i.e., what is the computational model)?

The basic computation that a general-purpose language performs is based on a computational model, which defines how the language processes and manipulates data.

The computational model used by most general-purpose languages is the Von Neumann architecture, which is named after the computer scientist John Von Neumann. This model consists of three key components: the central processing unit (CPU), memory, and input/output devices.

In this model, the CPU retrieves instructions and data from memory and performs arithmetic and logical operations on the data. The results of these operations are then stored back in memory for future use. Input/output devices, such as keyboards and displays, allow the user to interact with the program and provide input and output data.

General-purpose languages typically use high-level abstractions to simplify programming tasks for the developer, but ultimately the language's compiler or interpreter translates the program into machine code that can be executed by the CPU according to the Von Neumann architecture.

What are the basic data structures in your GPL?

The basic data structures are:

Arrays: An array is a collection of elements of the same type that are stored in contiguous memory locations. Arrays can be used to store and manipulate large amounts of data efficiently.

Lists: A list is a collection of elements that may be of different types and are stored sequentially. Lists can be used for dynamic data structures where the size of the data can change during runtime.

Sets: A set is a collection of unique elements that are unordered. Sets can be used to store and manipulate data that does not require a specific order.

Maps: A map is a collection of key-value pairs, where each key is associated with a value. Maps can be used to store and manipulate data where elements can be accessed based on a key.

Queues: A queue is a data structure that follows the first-in, first-out (FIFO) principle, where elements are added to the back of the queue and removed from the front.

Stacks: A stack is a data structure that follows the last-in, first-out (LIFO) principle, where elements are added and removed from the top of the stack.

How does the user create and manipulate data?

In a general-purpose language, users can create and manipulate data using a variety of constructs and functions provided by the language. Here is a general overview of how users can create and manipulate data in a general-purpose language:

Data types: The user can define variables and assign values to them based on the data types supported by the language. These data types may include integers, floating-point numbers, strings, and Boolean values.

Data structures: Users can create data structures like arrays, lists, sets, maps, queues, and stacks to organize and manipulate large sets of data.

Operators: Users can use operators provided by the language, such as arithmetic operators (+, -, *, /), comparison operators (==, !=, >, <), and logical operators (&&, ||) to perform various operations on data.

Functions: Users can define and use functions to manipulate data. Functions allow users to group related code into reusable blocks and can accept parameters and return values.

Control structures: Users can use control structures like if-else statements, loops (for, while, do-while), and switch statements to control the flow of execution and manipulate data based on conditions.

Libraries and frameworks: Users can use libraries and frameworks provided by the language or developed by third parties to access additional functionality and manipulate data in more complex ways.

Overall, users can create and manipulate data in a general-purpose language using a combination of data types, data structures, operators, functions, control structures, and libraries.

What are the basic control structures in your GPL?
Control structures are programming constructs that allow developers to control the flow of execution in a program. Some of the basic control structures that are commonly used in general-purpose languages include:

If-else statements: These statements allow developers to execute different code blocks based on a specific condition. If the condition is true, one code block is executed; otherwise, another code block is executed.

Loops: Loops allow developers to execute a code block repeatedly. The most common types of loops are:

For loops: These loops execute a code block a specific number of times based on a given range of values.

While loops: These loops execute a code block repeatedly as long as a specific condition is true.

Do-while loops: These loops execute a code block once before checking a specific condition, and then execute the code block repeatedly as long as the condition is true.

Switch statements: These statements allow developers to execute different code blocks based on the value of a variable. Switch statements are typically used with integer or string variables.

What kind(s) of input does a program in your GPL require?

In general, programs in a general-purpose language require the following types of input:

Command-line arguments: Programs can accept input arguments from the command line when they are executed. These arguments can be used to customize the behavior of the program, pass in filenames or other data, or specify options.

User input: Programs can prompt the user to enter input data while the program is running. This can be done through text prompts or graphical user interfaces (GUIs) depending on the type of program being developed.

Input files: Programs can read input data from files, either as command-line arguments or by prompting the user to select a file. The data can be read from a variety of file formats, including plain text files, JSON files, XML files, and binary files.

Data from other programs or APIs: Programs can access data from other programs or APIs, either by making network requests or by accessing data stored locally on the user's machine. For example, a program might access data from a database, a web service, or a file system.

What kind(s) of output does a program produce?

Text output: Programs can output text to the command line or to a file. This can include program output, error messages, or other status updates.

Graphical output: Programs can produce graphical output in the form of images, charts, or graphs. This can be useful for data visualization or for generating reports.

Audio output: Programs can produce audio output in the form of sound effects or music. This is particularly useful for games and other interactive applications.

Video output: Programs can produce video output in the form of animations or recorded video. This can be useful for video games or for applications that require video recording or playback.

Data output: Programs can output data to files or to other programs. This can be useful for data processing or for exchanging data between programs.

Error handling:

How might programs go wrong, and how might language communicate those errors to the user?

Programs in a general-purpose language can go wrong in many ways, such as logical errors, syntax errors, runtime errors, or hardware failures. Here are some common examples of errors that can occur in programs:

Syntax errors: These occur when the program code is not written correctly according to the syntax rules of the language. This can result in a program that does not compile or that produces unexpected behavior.

Logic errors: These occur when the program code does not produce the desired behavior, even though it may compile and run without error. These can be difficult to diagnose because the program may not fail outright, but instead produce unexpected results.

Runtime errors: These occur when a program is running and encounters an unexpected situation that it cannot handle. This can result in the program crashing or producing unexpected results.

Hardware failures: These occur when the hardware that the program is running on malfunctions. This can result in the program crashing or producing unexpected results.

When errors occur in a program, the language must communicate those errors to the user in a clear and understandable way. Here are some ways that a language can communicate errors to the user:

Error messages: When an error occurs, the language can display an error message that explains the problem and suggests possible solutions.

Debugging tools: Languages can provide tools for debugging programs, such as breakpoints, stack traces, and variable inspection, that allow developers to identify and fix errors in their code.

Exceptions: Languages can use exception handling mechanisms to allow programs to gracefully handle errors and recover from unexpected situations.

Logging: Programs can write logs that record errors and other important information during program execution. These logs can be used to diagnose problems and track down errors.

Are there any other GPLs for this domain?
If so, what are they, and how will your language compare to these other languages?

There are more general purpose languages in the world than there are stars in the sky like Python, C, C++, C#, JavaScript, Ruby, Kotlin, PHP, Go etc. So, those programming languages have been used for years so we will not be able to create a more attractive one.

Implementation plan:

How do the team plan to implement language?

There is the plan of implementation:

• Research GPL development steps. Frontend and Backend of compilers.

• Define Syntax of GPL

• Define Grammar and Rules

• Analyze LLVM (open source backend for compilers)

• Create Lexer

• Create Parser for generating AST

• Generate Intermediate representation for LLVM

• Build GPL by compiling intermediate representation using LLVM