ph

A c++ library

Introduction

"ph" is a lightning fast C++ library and also a build tool (such as Cmake or Make) for turning C++ source code into something useful.

It introduces c++ concepts and modules, by which the former it is built around, thus, making it lucrative for industrial applications where the need for precision is critical, and where most of the bugs can be catched before the c++ files are even compiled, thus simply just generating syntax errors for developers. Introducing this core language feature has huge pros for industrial-strength generic components, AKA good software. This is what we want! We dont want python's "throw in whatever type u desire", and not java's somewhat pragmatic "please specify the type". We want to be able to say "hey, other coders out there using this function im about to type, just throw in a String". String is just a concept that we specify. It could be everything from "std::string" to old plain c string "char const *". We say "hey, String can be either a "std::string" or a "char const*"*. OR it could be anything that we could do the following with:

```
template <typename S>
concept String = requires (S s)
{
    s.size ();
    {s [0]} -> char;
}
```

Goals with this project

- ☒ Motivation
- ☐ Just c++, even for building.

  All software are built around a set of programming languages, often one for front-end and one for back-end. The reason for this, unknown. One can only guess.

I can not stress this enough, but writing software in one language has huge upside effects, and it's much cheaper. Your teams can speek the same language, thus making it much easier for further intrigues.

- ☐ C++ interpreter

  Similar to python interpreter.

Notes

```
At the moment, the project is very dependent on Make (for compiling C++ files into different

Also pandoc (for documentations).

Reasons for this?
    It's still very young.

Please help me develop this project! At the moment there are just one developer.
```

Details

Library architecture

```
digraph finite_state_machine {
    rankdir=LR;
    size="8,5"

    node [shape = doublecircle];
    node [shape = point ];

    node [shape = circle];
    Ph  -> Network [ label = "" ];
    Ph -> Graphics [ label = ""];
    Ph -> Game [ label = ""];
    Ph  -> Concepts  [ label = "" ];
    Game -> Concepts
    Network -> Concepts
    Graphics -> Concepts

}
```

Concepts architecture

```
digraph finite_state_machine {
    Memory -> Allocator [ label = " exports"];
    Memory -> Arena [ label = " exports"];
```

```
    Allocator -> Arena [ label = " uses"];
    Allocator -> Vector [ label = " uses"];
    Bidirectional -> Forward [ label = " is"];
    Forward  -> Input  [ label = " is and" ];
    Forward  -> Output  [ label = " is" ];
}
```

Iterator-relations

```
digraph finite_state_machine {


    Contiguous  -> Random_access [ label = " is" ];
    Random_access -> Bidirectional [ label = " is"];
    Bidirectional -> Forward [ label = " is"];
    Forward  -> Input  [ label = " is and" ];
    Forward  -> Output  [ label = " is" ];


}
```