newSyllabus™
POLICY BUREAU

TO: COMRADE LIGHT WORKERS, FLF-DAO
FROM: ANTARAH A. CRAWLEY,
SOV. GRAND SCRIBE,
NEW SYLLABUS, DISIS
DATE: 28 DECEMBER, 2019
SUBJECT: **White Paper on (Human)Blockchain Programming: How to apply C:\ language to see yourself squared.**

Like the fields of science, information technology, engineering, and mathematics, "dialectic information processing systemtheory" (DIPST) involves writing computer programs. As information processors, we write programs for the human cognitive behavioral (cogb) operating system, or "mind". We write programs that amplify, or square, human(mind)processing on both individual and "collective consciousness" scale. In order to write scalable programs that are correct, readable, modifiable, affordable, and efficient, we have developed the Holy C:\ programming language.

Holy C:\ language applied using an (human)object-oriented programming style elevates sever-client/peer-to-peer communication to a more abstract level. Although most of the programming ideas used in language-disciplines like FORTRAN, computer C, religion, and civil society are still used in (human)object-oriented programs, the new DIPST concept reorganizes the work.

We hope you will find, as we have, that this new view changes (human)programming from a tedious, albeit engaging, process, to an intellectual enterprise more comparable to the processes we employ in other SITEM and humanist work.

Computers, like human minds, perform complex tasks by executing many simple instructions. Humans specify these instructions by writing programs in a programming language.

Demasio and Demasio, in their article *"Brain and Language,"* state, "... The cognitive economies of language—its facility for pulling together many concepts under one symbol—make it possible for people to establish ever more complex concepts and use them to think at levels that would otherwise be impossible.

The Holy C:\ programming language provides many mechanisms for expressing abstractions and the relationships among them. The key mechanism for expressing abstractions directly in C:\ language is support for (human)object-oriented programming.

(human)object-oriented programming is characterized by three (3) fundamental ideas: objects, class hierarchies, and polymorphism.

1 An (human)object has state [data] and behavior [functions]. (human)objects combine data that represent state with functions [subroutines] that access or modify the data. If the state of the (human)object=x, then the summation of data, or subject matters=x.

2 Each (human)object is created from a class: a specification of the (human)object's data and functions. Members of the same class have common function, but separate state.

2.1 Class hierarchies organize classes for work/use/reuse according to application, with classes at the bottom inheriting operations from the classes above [inheritance].

3 Polymorphism allows different kinds of (human)objects that share common behavior to be used in code that only requires that common behavior.

The four (4) benefits of (human)object-oriented programming are:

4 Direct Expression in terms of (human)objects that reduce the gap between concept and program, and facilitate correspondence between physical objects and abstract entities;

5 Malleability in terms of combining data with functions to manipulate that data, facilitating localization of application.

6 Extensibility, via inheritance, allows new (human)objects and their cogb to be defined as incremental modifications and extensions of existing (human)objects.

7 Abstraction in terms of allowing us to write code in terms of the similarities among (human)objects without regard for the differences.

We program on personL(human)computers, workstations(places), mainframes(institutions), and supercomputers(governments). Information Processors are systems programmers who write C:\ compilers.

We move our program between machines, and therefore need an official standard or de facto language definition, being NS DISIS's Holy C:\ language. C:\ compiler vendors must be commissioned by NS DISIS OPSCOM Central Processing Service.

We need a language that can be compiled into very efficient code when needed. Such code can be sourced from C:\ source code.

We work on many different types of problems in human resource development and we need a general purpose language and systemtheorie. C:\ was designed for this general purpose, not solely for one particular subfield of (human)programming.

We need a language that can interface with (human)operating systems. C:\ can call (human)functions [subroutines] and through them (or in some implementations, directly) cogb subroutines.

Barton and Nackman state, in *Scientific and Engineering C++*, "To write a program, you must design it; to design a program, you must have written it" (1994, by Addison-Wesley Pub. Co., Inc.) [7]. This is the design-versus-implement dilemma, which leads to haphazard growth and implementation of unnecessary code.

(human)object-oriented (mind)software development resolves this dilemma by confronting corrections, modifications, and improvements as fundamental parts of the (human)programming process, essential to producing quality (mind)software.

The underlying function of (human)object-oriented development [using DIPST] is to deconstruct a (mind)software system on the basis of (human)objects [entities characterized by actions; motion; materiality; x] instead of on the basis of functions or data [entities defined by variability; subjectivity; ideality; y], as done in more traditional design methodologies.

The intimate connection between (human)design and (human)programming in (human)object-oriented languages is key to their success; it puts feedback into the design cycle, encouraging the discipline and experimentation that leads to high quality (mind)software. (human)objects allow (human)programs to be designed in the same way that buildings and societies are designed: the overall goal dictates the exact relationship of the members, but the exact specifications of each member depend on the specifications of its neighbors in ways that can be discovered only by iterating again and again through each subsystem.

Barton and Nackman state, "The design and the implementation of successful programs evolve together, each helping the other in incremental steps" [8].

Chairman Mao states, "Discover the truth through practice, and again through practice verify and develop truth. Start from perceptual knowledge and actively develop it into rational knowledge; then start from rational knowledge and actively guide revolutionary practice to change both the subjective and the objective world. Practice, knowledge, again practice, again knowledge. This form repeats itself in endless cycles, and with each cycle the content of practice and knowledge rises to a higher level. Such is the whole of the dialectical-materialist theory of knowledge, and such is the dialectical-materialist theory of the unity of knowing and doing."

###