**Beyond C++: проект современного языка программирования общего назначения**

**1. Предпосылки**

Язык программирования, со своей инфраструктурой – системами поддержки времени выполнения, стандартными и прикладными библиотеками, инструментами и средами разработки (компиляторы, редакторы связей, IDE) представляет собой ключевой инструмент создания программного обеспечения и, тем самым, служит определяющим фактором обеспечения эффективного, безопасного и надежного функционирования многочисленных и разнообразных электронных устройств в современном мире.

Существующее положение в сфере языков программирования и, шире, в области инструментария современного программирования весьма далеко от идеала. Большинство языков программирования, наиболее широко используемых в настоящее время, создано более двадцати лет назад и в настоящее время совершенно не адекватно практически ни одному из современных требований, предъявляемых к инструментам разработки современного ПО. Эти языки архаичны, неуклюжи, громоздки, неудобны и сложны в практическом использовании, не способствуют надежности и эффективности создаваемого ПО, зачастую несут явный отпечаток вкусовых пристрастий и причудливых взглядов их создателей.

Новые языки программирования, в изобилии появляющиеся в последние пять-семь лет, пытаются преодолеть указанные недостатки, однако в значительной степени повторяют устаревшие и порочные подходы проектирования, берущие своё начало в восьмидесятых годах прошлого столетия.

**2. Существо проекта**

Основной смысл предлагаемого проекта заключается в том, чтобы спроектировать и реализовать оригинальный язык программирования (предварительное название – **SLang**) вместе с необходимой экосистемой: компилятором, подсистемой поддержкой времени выполнения, стандартными библиотеками, редактором связей (комплексатором).

Основной смысл проекта заключается в том, чтобы предложить разработчикам программного обеспечения XXI века инструмент, который бы позволил им решать разнообразные задачи наиболее простым и надежным способом в соответствии с предъявляемым требованиям и применительно к разнообразным уровням квалификации разработчиков.

Язык SLang воплощает адекватное понимание существа процесса проектирования и разработки современного ПО. Он включает свойства, обеспечивающие надежность, безопасность и эффективность программ, создаваемых с его использованием. В то же время он достаточно прост для обучения, освоения и использования, что обеспечит «гладкий» процесс разработки и сопровождения, а также предоставит возможность включить в сферу разработки ПО более широкие, нежели в настоящее время, сообщества разработчиков, в том числе, непрофессионалов.

Язык SLang носит принципиально универсальный характер, и потому пригоден для успешного создания ПО в различных сферах применений – от микроустройств с минимальными характеристиками производительности и памяти (что особенно существенно в сфере «интернета вещей», IoT), мобильных устройств, приложений для сети Интернет, до серверных систем, суперкомпьютеров и систем реального времени.

Язык SLang – *масштабируемый*; под этим подразумевается его пригодность для реализации программных систем различного масштаба и сложности: от многофункциональных, логически насыщенных систем с повышенными требованиями к надежности, до коротких программ («скриптов»), решающих простые прикладные задачи.

Помимо языка, проект будет включать необходимый (на первом этапе – минимальный, в итоге – исчерпывающий) набор инструментов, в совокупности обеспечивающих полный жизненный цикл программ.

Язык спроектирован как *машинно-независимый*. Его семантика имеет высокий уровень и не ориентирована на особенности какой-либо аппаратной или программной платформы. В то же время, язык может быть эффективно реализован для любой распространённой в настоящее время среды. *К ним относятся Linux и её наиболее популярные клоны, версии и сборки, iOS (Apple), Windows и Windows Phone, а также специализированные среды, в том числе, встроенные системы. (Вот это предложение можно удалить при сокрашении – зачем перечислять то что на слуху)*

*Замечание об импортозамещении.* В настоящее время достаточно остро стоит вопрос создания отечественных версий программного обеспечения различного рода. *Речь идет о полном спектре программных средств – от программ прикладного характера до базового, системного ПО – операционных систем, систем управления базами данных (СУБД), встроенных систем специального назначения. (Вот это предложение можно удалить при сокрашении – зачем перечислять )*

Для того чтобы импортозамещние в сфере разработки программного обеспечения было полным необходим также и современный отечественный язык программирования и все необходимые для разработки программного обеспечния инструменты.

*Не вдаваясь в обсуждение нетехнических аспектов этой тенденции, следует заметить, что общей проблемой, связанной с информационной независимостью государства, является вопрос инструментальных средств, используемых для построения отечественного ПО. В самом деле, программная система, полностью разработанная силами отечественных специалистов, чтобы превратиться в работающую программу, должна быть преобразована в машинное представление, подлежащее выполнению на некотором аппаратном оборудовании. Такое преобразование производится посредством специальных инструментов – компиляторов языков программирования, компоновщиков (редакторов связей), статических анализаторов программ и т.д. Представляется очевидным, что полная информационная независимость может быть достигнута только в случае создания и применения отечественных инструментальных средств. (И тогда весь этот абзац сократить)*

*К сожалению, этот, на наш взгляд, ключевой аспект зачастую оставляется без внимания. По нашему мнению, усилия, предпринимаемые в направлении создания отечественного ПО, должны в обязательном порядке подразумевать разработку отечественных инструментальных средств.* Проект языка программирования SLang и системы программирования на его основе призван заполнить существующий пробел и предложить мощное и современное решение в указанной сфере.

**3. Краткая характеристика языка SLang**

***Модульность*.** В отличие от многих современных языков, программа на SLang формируется по модульному принципу: строительными блоками любой программы служат *модули* – независимо определяемые, независимо хранимые и независимо компилируемые единицы со строго определенными интерфейсами, согласно которым они могут вступать в различные отношения друг с другом: использование, агрегация, наследование и т.д.

В языке определены два основных вида модулей: контейнеры и подпрограммы. Контейнеры представляют агрегацию логически связанных ресурсов (данных и подпрограмм-членов), подпрограммы реализуют некоторую функциональность и, в свою очередь, могут представлять собой процедуры или функции.

Продолжая эту логику, естественно считать, что компонент любого из указанных видов может служить *единицей компиляции*, то есть, допускать раздельную компиляцию. Отсутствие ограничений способствует созданию композиции программы, адекватной требованиям и особенностям ее использования. Программа может представлять собой, по сути, произвольную композицию единиц, от простого набора взаимодействующих подпрограмм до сложной комбинации контейнеров различных видов.

В предельном случае единица компиляции или вся программа может представлять собой единственную единицу – простую последовательность операторов. Если необходимо написать несколько строк кода, которые будут служить реакцией на нажатие клавиши мыши в некотором средстве просмотра Интернет-страниц, то нет необходимости писать подпрограмму – достаточно просто задать последовательность операторов, которая выполнит нужное действие. Если же решение задачи предполагает более сложную логику, то результат можно получить, комбинируя отдельные подпрограммы, контейнеры и блоки. Таким образом, единая языковая нотация может быть использована для решения максимально широкого класса задач.

**Строгая типизация.** Понятие типа является одним из базовых понятий любого языка. Под типом некоторого объекта, существующего в программе, понимается тройственная сущность, определяемая множеством значений, которые может принимать данный объект, связанным с ним множеством операций, допустимых над значениями данного типа, а также множеством отношений между данным типом и другими типами.

Язык SLang представляет собой язык со статической типизацией объектов. Это означает, что тип является неизменным свойством объекта; это свойство присуще объекту с момента его возникновения в программе и не может измениться во время жизни этого объекта. В терминах программирования это означает, что тип объекта назначается объекту (явно или неявно) при его объявлении, и не существует возможностей (языковых конструкций), позволяющих изменить тип в процессе выполнения программы.

Тип может быть (явно) приписан объекту программистом при объявлении объекта, либо (неявно) выведен компилятором из контекста объявления этого объекта. Примером контекста в данном случае может служить тип инициализирующего выражения из объявления объекта. (Если сжимать то текст курсивом модно упросить – по сути можно просто сказать строгая типизация + выведение типов где возможно – вот собственно говоря и все ☺)

Текущее значение объекта может быть преобразовано в значение другого типа. Правила преобразования типов задаются в явном виде при определении реализации типа программистом. НЕТ У НАС ЯВНЫХ ИЛИ НЕЯВНЫХ КОНСРУКЦИЙ ПРЕОБРАЗОВАНИЯ ТИПОВ!!! Мы просто переопределяем процедуру с именем := или задаем setter := для атрибута … ***Поддержка различных парадигм программирования*.** SLang – мультипарадигменный язык, в том смысле, что в нём воплощены важнейшие современные концепции программирования, включая объектно-ориентированное, обобщённое (generic) и функциональное программирование.

В языке имеется понятие типа (класса), которое реализуется посредством языковой конструкции «контейнер» (см. ниже), со всеми традиционными свойствами – инкапсуляцией , наследованием, полиморфизмом. Имеется возможность задавать абстрактные классы, а также реализовывать между классами отношения множественного наследования. При чем язык предлагет оригинальный подход к поддержке множественного наследования с возможностью конфликтов и с правилами их разрешений.

Любой контейнер или подпрограмма может быть параметризована типом (типами) и/или константой (константами). Настройка обобщенных программных единиц предполагает задание конкретных типов и/или значений. Тем самым, в языке обеспечивается полная поддержка парадигмы обобщенного программирования, что позволяет проектировать компоненты программы в виде, максимально независимом от контекстов их использования.

В языке поддерживается необходимый набор средств функционального подхода к программированию, включая функциональные типы, лямбда-выражения и замыкания. Эти свойства основаны на трактовке функций как значений, а также предполагают свободное использование понятия неизменяемых (immutable) объектов. ЧТО ТАКОЕ immutable object??? Или это ты так назвал константные объекты?

***Контейнер: модуль, класс и тип в одном флаконе*.** Важнейшими концепциями, используемыми при разработке программного обеспечения (ПО), служат понятия атрибутов (данных) и подпрограмм (действий). Атрибуты могут изменяться подпрограммами в процессе работы программы; они образуют ее вычислительный контекст, в то время как подпрограммы задают алгоритм решения задачи. Между атрибутами и подпрограммами есть логические связи, и объединяя атрибуты и подпрограммы в единый именованный контейнер, мы просто фиксируем эту связь. Таким образом, понятие *контейнера* (английский термин, выбранный для его наименования, – unit) можно считать простым средством агрегации логически связанных данных и действий в единое целое.

Более строго, *контейнер* (*unit*) можно определить как поименованную совокупность атрибутов и подпрограмм, которая может быть параметризована либо типами, либо константами, и может быть использована для задания типов, конструирования новых контейнеров при помощи наследования или для прямого использования атрибутов и подпрограмм данного контейнера в других контейнерах и отдельно-стоящих подпрограммах.

Контейнер можно рассматривать как определение множества данных и операций над ними – то есть, как задание некоторого *типа*. Тем самым, можно определить объект, тип которого будет контейнером. Во-вторых, можно предоставить открытое (общедоступное) содержимое контейнера для *использования* в некотором программном коде, то есть, включить его ресурсы в некоторый контекст (это схема использования модулей). Наконец, атрибуты и подпрограммы контейнера могут (пере)использоваться при создании нового контейнера. Такой механизм носит название *наследования*.

Таким образом, различные варианты использования контейнера приводят к понятиям *типа*, *модуля* и *класса*. В языке SLang сохраняются преимущества единой нотации задания контейнеров, с возможностью явного задания *различных способов использования* контейнеров.

***Однородная система типов*.** Система типов языка SLang является *однородной*. Это означает, что в языке отсутствует деление на различные категории типов (например, «встроенные в язык» и «определяемые пользователем»). Любой тип, используемый в программе, определяется единообразно, посредством универсальной конструкции «контейнер» (unit). Единственное различие в системе типов заключается в том, что некоторые наиболее часто используемые на практике типы – целый, вещественный, логический, а также такие структуры данных, как массивы, списки, словари и т.д. – являются частью библиотеки контейнеров, которая поставляется вместе с компилятором и часть описаний атрибутов и подпрограмм таких контейнеров зафиксированы. *Понятие контейнера предоставляет удобное и универсальное средство создания разработчиком новых типов на основе существующих. При этом допускаются все традиционные методы и практики определения типов, в том числе агрегирование и наследование.*

*Возможность создавать и использовать в программе некоторый новый тип – необходимое условие гибкости языка и, как следствие, залог его возможностей по адекватной реализации множества реальных структур данных и отношений между ними. Использование существующих типов в качестве составных частей других типов в значительной степени обеспечивает мощность языка и выразительность программ, написанных на этом языке.*

*Однородность системы типов существенно упрощает понятийный базис языка, делая его стройным, логичным, простым для понимания, тем самым обеспечивая максимально возможную простоту, ясность, недвусмысленность и надёжность программ. В то же время, однородность не ограничивает выразительные возможности языка, предоставляя в распоряжение разработчика полный спектр инструментов для создания структур данных любой сложности. (кусок выделенный курсивом – можно сократить. Тут много просто общих моментов )*

***Контрактное программирование***. Подход к проектированию программ на основе понятия *контракта* (Design by contract © ), изначально разработанный и обоснованный Б.Майером и реализованный в языке Эйфель, в настоящее время является общепринятым средством повышения надежности и верифицируемости ПО. Подход в том или ином объеме реализован во многих современных ЯП. Язык SLang поддерживает полный спектр механизмов *контрактного программирования*, включая пред- и постусловия для подпрограмм и инварианты контейнеров и циклов. Система поддержки времени выполнения обеспечивает эффективную (параллельную, если это возможно) проверку предикатов.

***Параллельное программирование***. В отличие от большинства современных языков, где поддержка параллельности реализована на уровне библиотек и носит ограниченный и слабо верифицируемый характер, SLang включает удобный и достаточно надежный механизм распараллеливания программ на уровне самого языка. В языке имеется простой и компактный набор конструкций и спецификаторов для задания параллельного выполнения и синхронизации по доступу к данным. А также компилятор должен производить автопараллелизацию, где возможно.

***Безопасность*.** Проблема, связанная с неконтролируемым использованием нулевых указателей («пустых» или «повисших» ссылок), является одной из наиболее распространенных в практике программирования, а также одной из самых опасных по своим последствиям с точки зрения обеспечения надежности программ. В то же время, контроль доступа по таким указателям не имеет удовлетворительного решения в традиционных языках.

В языке SLang проблема пустых указателей трактуется не как самостоятельная проблема, а как часть более общей проблемы некорректной работы с *неинициализированными атрибутами*. Пустая ссылка считается разновидностью неинициализированного атрибута, и в языке имеются механизмы, которые строго ограничивают случаи, когда действительно нужны неинициализированные атрибуты, от ситуаций, когда всякая сущность должна иметь определенное значение. В дополнение к этому имеется надежный механизм перехода от потенциально неинициализированных атрибутов к инициализированным – своего рода мостик от «опасного» мира в «безопасный». И эта схема работает как для ссылочных типов, так и для типов значений.