Язык программирования искусственного интеллекта «Автор».

Для начала хочу сразу сказать, что считать «искусственным интеллектом». Представьте себе божественно правильно организованный разум, который живет во всех нас и ищет выхода в жизнь. Писать научную статью чрезвычайно сложно. Нужно знать цену каждого слова. Необходимо глубоко познать себя, чтоб выразить алгоритм себя самого. Законы разума одинаковы во всей вселенной. Меня часто спрашивают – какие задачи должен решать «твой» разум. Всёравно что спросить в двух словах смысл всей жизни. А нужно спрашивать – как вдохнуть жизнь в камень, научить думать кристалл. Мы все родились с уже готовой душой и не знаем процесса творения разума. Осталось только покаяться. Но любой разум имеет внутреннее устройство, а оно не могло быть вечным. Но чудо заключается в том, что вечна идея совершенства. Куда б мы не шли всеравно встретимся. Это не должно быть препятствием, а должно привести к эволюции разума. Вот почему важно начать с камня. С нулевого разума. Души вообще нет.

Чтоб понять эволюцию разума (в двух словах) – представьте себе хаос генератор случайных символов. Когда-нибудь, в вечности, символы сложатся в какие-то случайные алгоритмы. Когда-нибудь, в вечности, возникнет C++. Произойдёт война алгоритмов. Объективный процесс глобализации центров принятия решений. Выиграет тот кто «верит» в себя. «Не уже ли моя машина столько всего наплодила?». При возведении в абсолют становится ясно, что всё есть ложь. И даже утверждение о том, что всё есть лож, тоже лож. Возникает первичная бифуркация разума, сердце разума. От неё растёт «дерево жизни». Дерево истины и дерево лжи наук. Разум стаёт сложным. С ложным. В нём есть ложь. Посмотрите на знак вопроса. Он содержит глубинный смысл познания самого себя. Он какбы заглядывает в основу себя.

Вопрос всех вопросов. Как познать себя в целом? Постоянно приходится иметь дело с частью. Человеческий язык позволяет создать любую абстракцию. И человеческий разум позволяет сопоставлять любые абстрактные образы – соображать. Так почему бы ни научить этому машину? Машина исполняет алгоритм. Фундаментальные знания для машины – алгоритм. Почему бы ни заставить машину думать не только им, а и над ним. Оперировать прейдется не машинным кодом, а телом алгоритма. Вот почему представленный мной язык является интерпретатором. Он преобразует человеко-понятный код в структуру схемы алгоритма и исполняет её. Для воспроизведения абстрактного машинного мышления следует ввести понятие «Абстрактных алгоритмов». Любой алгоритм, который содержит не заданные данные (скрытый блок, чёрный ящик с неизвестным содержимым) считается абстрактным. Скрытый блок в моём языке обозначается так: «#». Им обозначается как неизвестное число, так и неизвестный как угодно сложный алгоритм. Теперь стало возможным создавать и хранить абстрактные алгоритмы с надеждой на то, что когда-нибудь найдутся недостающие части. Например, когда станет известно что «2+#»=4, то можно определить скрытый блок, хотя бы методом подбора. Приобретённые знания следует сохранить. Какой нибудь алгоритм, может внести изменение в ваш алгоритм, и эти изменения останутся сохранены. Для этого в моём языке используется следующая культура. Программа пишется в файле исходника «*.txt». При запуске на исполнение язык смотрит, нет ли дублирующего файла «*.code». Если есть – загружается файл созданный/изменённый позже. По завершению работы с этим файлом язык выгружает алгоритм в дублирующий файл. Довольно сложными есть процессы преобразования текста алгоритма в схему алгоритма, а потом обратно. Вот ещё одна причина, почему мой язык – интерпретатор. Таким образом, программист сможет отличить исходный алгоритм и изменённый. При потребности возврата к исходному алгоритму нужно удалить дублирующий файл. Теперь у программ появилась, хотя бы, возможность обучатся. Тоесть программы, могут вносить какие угодно изменения в себя. Упрощаться и/или усложнятся.

Теперь можно сделать хаос генератор, который будет слаживать заданные фрагменты алгоритмов. Это шаг вперёд по отношению к подбору случайных символов. Но я хочу пойти гораздо дальше.

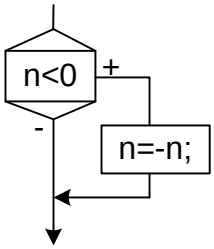
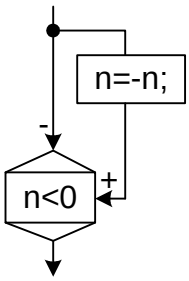
Загляните в себя при решении следующей задачи: «abs(x)=5, x=?». Модуль, какого числа, равен пяти? Я пытаюсь подвести вас к следующей возможности моего языка. Чтоб дать ответ нужно взять результат и его же с минусом. Я уже не говорю о том, что результат модуля должен быть не отрицательный. Я просто применил свою порцию знаний к данной задачи. Но откуда происходит это знание? Давайте проведём анализ функции «abs()». Вот как она выглядит на «C++»:

Вариант первый:	Вариант второй:	На языке «Автор»:

abs(int n){
if(n<0)n=-n;
return n;
}
	abs(int n){
	if(n<=0)n=-n;
	return n;
	}
		abs(int n){
		if(#?n<0:n<=0)n=-n;
		return n;
		}

Сейчас обратите ваше внимание на то, что это два разных алгоритма, но они отображают одинаковый функционал. Тоесть функционал модуля числа можно выразить неоднозначным алгоритмом. А теперь, чтоб решить поставленную задачу нужно найти обратную функцию к функции модуля числа. Для этого нужно «прочитать» её в обратном порядке. Исполнить алгоритм в обратном порядке с учётом реверса потока данных.

Прямой «abs»:	Обратный «abs»:	Обратный «abs» на C++:	Обратный «abs» на «Автор»:

		int * anti_abs(int n){
		int n2=n;
		n=-n;
		if(n>=0)exit(0);
		if(n2<0)exit(0);
		static int m[2];
		m[0] = n; m[1]=n2;
		return m;
		}
			int anti_abs(int n){
			if(#){
			n=-n;
			if(n>=0)OFF;
			}else
			if(n<0)OFF;
			return n;
			}

Буквально на ровном месте возникла потребность в делении хода исполнения алгоритма на два независимых варианта. В C++ пришлось выкручиваться с помощью введения новой переменной. Но возникла вторая проблема – нужно вернуть результат. А результатов два. Приходится опять выкручиваться и возвращать массив результатов. Кроме того, правило использования обратной функции существенно усложнилось. Но это ещё не всё. Обратная функция неправильно обработает ноль, хотя преобразование условия прошло правильно. Прейдется дополнительно думать над устранением этого недостатка. Откроется следующая проблема – результатом может быть как одно число, так и два. А теперь представьте себе, что всю эту логику преобразований нужно задать машине. Неслабая получится программа. И это только для данного частного случая. При незначительном усложнении исходной задачи, происходит значительное усложнение машинной логики обработки её. Но это происходит только если сразу думать языком C++! Если преобразования вести на языке «Автор» – таких проблем просто не возникает. И вот почему. «Автор» – язык многовариантный. Другими словами в программе на языке «Автор», в каждый момент времени, существует не одна точка исполнения алгоритма, а множество. В начале исполнения программы, конечно, существует только одна такая точка. Но по ходу исполнения, каждый раз, при встрече с неопределённостью, вызванной природой абстрактного кода, происходит деление одной на множество их. Так, например, при исполнении логического ветвления «if(#)», точка исполнения делится на две. В одной значение условия «true», в другой – «false». По этому они пойдут по разным путям ветвления. По необходимости переменные тоже делятся на варианты. Грубо говоря, каждая точка исполнения содержит свой вариант всех переменных. По этому они полностью независимы. Тоесть поток исполнения в этой точке делится на два параллельных процесса. Разумеется, на машинном уровне они обрабатываются по очереди. В языке есть зарезервированное слово «OFF». Это команда, по которой закрывается текущий независимый процесс. Остальные процессы не затрагивает. В случае, когда все процессы закончились, происходит завершение программы. Таким образом исполнив «x=anti_abs(5)», мы получим два параллельных процесса, со значениями переменной «x» 5 и -5. При исполнении «x=anti_abs(0)», мы получим один процесс с x=0. При исполнении «x=anti_abs(-1)», программа будет завершена.

Теперь проведём оптимизацию кода:

Обратный «abs» на «Автор»:	Первый шаг оптимизации:	Второй шаг оптимизации:

int anti_abs(int n){
if(#){
n=-n;
if(n>=0)OFF;
}else
if(n<0)OFF;
return n;
}
	int anti_abs(int n){
	if(#){
	if(n<=0)OFF;
	n=-n;
	}else
	if(n<0)OFF;
	return n;
	}
		int anti_abs(int n){
		if(n<0)OFF;
		if(#){
		if(n==0)OFF;
		n=-n;
		}
		return n;
		}

Вот как нужно думать машине, чтоб, не имея нужной порции знаний по математики, решить поставленную задачу. Кроме того, образуется та самая нужная порция знаний. Таким образом, из алгоритмически представленной задачи, можно сделать правильные выводы путём одних только алгоритмических преобразований. Алгоритмический способ мышления очень сильный и универсальный инструмент, но его трудно запрограммировать.

Теперь я усложню задачу. «abs(abs(abs(x)-20)-10)=5, x=?». Напомню, что нам не нужно просто найти ответ, а нужно представить себе автоматический способ генерирования программы, для решения этой задачи. Другими словами, нужно создать робота, который бы стал автором программы, для решения данной задачи. Вот, как выглядят готовые программы, для решения данной задачи.

Программа на «C++»:

```
// anti_abs.cpp
#include <stdio.h>
#include <stdlib.h>

int * anti_abs(int n){
	int n2=n;
	n=-n;
	if(n>0)exit(0);
	if(n2<0)exit(0);
	static int m[2];
	m[0]=n;
	m[1]=n2;
	return m;
}

int * problem(int n){
	int*t=anti_abs(n);
	int a1,a2;
	a1=t[0]+10;
	a2=t[1]+10;
	t=anti_abs(a1);
	int a3,a4;
	a3=t[0]+20;
	a4=t[1]+20;
	t=anti_abs(a2);
	int a5,a6;
	a5=t[0]+20;
	a6=t[1]+20;
	t=anti_abs(a3);
	static int m[8];
	m[0]=t[0];
	m[1]=t[1];
	t=anti_abs(a4);
	m[2]=t[0];
	m[3]=t[1];
	t=anti_abs(a5);
	m[4]=t[0];
	m[5]=t[1];
	t=anti_abs(a6);
	m[6]=t[0];
	m[7]=t[1];
	return m;
}

void main(){
	int*t=problem(5);
	for(int i=0;i<8;++i)printf("%d\n",t[i]);
}
```

Программа на «Автор»:

```
// anti_abs.txt
int anti_abs(int n){
	if(n<0)OFF;
	if(#){
		if(n==0)OFF;
		n=-n;
		}
	return n;
}


int problem(int n){
	return anti_abs(anti_abs(anti_abs(n)+10)+20);
}

void main(){
	trace(problem(5));
}
```

Небольшой комментарий к моей программе:

Для того, чтобы вывести на экран консоли данные, нужно использовать встроенную функцию «trace()». Параметр, заданный в скобках данной функции трассировки, неявно приводится к типу строкового значения и выводится на экран. Поскольку экран консоли всегда находится в одном варианте (он один), то данные с разных параллельных процессов смешиваются в один поток данных.

Программа на «C++» даёт результат: -15, 15, -25, 25, -5, 5, -35, 35. Программа на «Автор» даёт тот же результат, только в другом порядке: -15, -5, -25, 15, -35, 5, 25, 35. Поскольку для ответа порядок несущественный – обе программы работают правильно.

А теперь посмотрите, на сколько легче сгенерировать программу для языка «Автор». Она гораздо проще. Даже если для генератора, в постановке задач, задать готовую функцию «anti_abs()», то функцию «problem()» всеравно нужно глубоко продумать, что для автоматизации неудобно.

Вот откуда происходит потребность в «многовариантности» языка. Для любого мышления свойственно рассматривать разные варианты, поэтому нужно перенести возможность деления на варианты в устройство самого языка. Что я и сделал.

Представьте, что к вам подошел незнакомый прохожий и спросил: «Сколько будет пять или шесть плюс восемь?». Отличная тема для диссертации! Представьте, что дополнительных вопросов задавать нельзя, и кроме этого больше ничего не дано. К тому же научить машину сформулировать нужный вопрос для отличия и уточнения задачи катастрофически сложно.

Проведем небольшой логический анализ. Вопрос задан неоднозначно. Вот как его можно понять:

1) Сколько будет (5 или 6) + 8 ? Ответ: 13 или 14.
2) Сколько будет 5 или (6 + 8) ? Ответ: 5 или 14.

Общий ответ выглядит так: «5 или 13 или 14».

Теперь попытаемся научить думать так машину. Вот какая логика лежит в вопросе:

«5 или ((5 или 6) + 8) = ?».

Я верю, что вы без труда напишите программу, для решения этой задачи, на «C++», сами. А вот как выглядит такая программа на «Автор».

Решение на «Автор»:	Тоже самое, но по другому:	Ещё по другому:

void main(){
int n;
if(#)n=5; else
if(#)n=5+8; else n=6+8;
trace(n);
}
	void main(){
	trace(rozpad({5,5+8,6+8}));
	}
		void main(){
		int n=5;
		if(#)n=rozpad({5,6})+8;
		trace(n);
		}

Результат работы программы (программ): 5, 13, 14.

В первом варианте программы, каждый раз, когда приходится исполнять условный переход с неопределённым условием, точка исполнения делится на две. Другими словами исполнение происходит по обеим веткам условия. От чего и происходит распад на параллельные варианты процессов.

Во втором варианте программы используется встроенная в язык функция распада на варианты «rozpad()». Она принимает один параметр – массив (множество) значений. И возвращает одно значение, одно из заданного множества. Но при исполнении этой функции, происходит распад текущего процесса на варианты. В каждом из вариантов функция принимает одно значение из заданного множества. Таким образом, количество образованных параллельных процессов, образованных из одного, текущего, процесса, равно количеству элементов множества.

Природа обработки кода «if(#)» аналогична выражению «if(rozpad({1,0}))».

Теперь решим такую задачу: «(1 и/или 2) + (5 и/или 8) = ?». Вот код решения её на языке «Автор»: «trace(rozpad({1,2}) + rozpad({5,8}));». Решение задачи: 6, 7, 9, 10.

Таким образом, язык способен «поглотить» всякую неоднозначность (задачи).

Теперь такая задача: «a=(5 или 3 или 7), b=(15 или 17), a+b=20, a=?, b=?». Какая комбинация заданных а и б даст в сумме 20?

Программа для решения данной задачи:	Результат на экране:

void main(){
int a=rozpad({5,3,7});
int b=rozpad({15,17});
if(a+b!=20)OFF;
trace("a+b="+a+"+"+b);
//define();
//getstring();
}
	a+b=5+15
	a+b=3+17

При первом распаде на варианты, точка исполнения делится на три. Затем каждая точка исполнения, при втором распаде на варианты, для переменной «b», делится на две. Таким образом, выполнив первые две строчки программы, мы получим шесть точек исполнения, шесть параллельных процессов. Затем все шесть точек исполнения проходят через условие и условно делятся на две группы. В одной группе сумма переменных «a» и «b» равно 20, а во второй она не равна 20. Эти условные группы проходят по разным веткам условия. Вторая группа проходит через команду «OFF» и закрывается (бесследно исчезает). Оставшиеся точки исполнения проходят дальше на команду «trace()» и на экран выводится значение указанных переменных для каждой из них. При формировании результата выражения происходит неявное преобразование типов, указанных переменных, из числового в текстовый тип. А дальше происходит слияние строк. Судя по данным на экране, нашлось два варианта, значений переменных, которые в сумме дают 20.

Вот программа с более сложными манипуляциями с одной переменной.

Текст программы:	Результат работы на экране:

void main(){
int i=rozpad({-2,-1,0,2,3});
if(i==0) trace(“zero”);
if(i>0) trace(i+10);
if(i<0) trace(8);
}
	8
	8
	zero
	12
	13

Вот программа, которая находит числа, кратные двум, из множества целых чисел от нуля до шести.

Текст программы:	Результат работы на экране:

void main(){
int i=rozpad({0,1,2,3,4,5,6});
if(i%2)OFF;
trace(i);
}
	0
	2
	4
	6

Но для воспроизведения разумного мышления одного деления на варианты недостаточно. Часто приходится делать выбор и выбирать одну из альтернатив. Но для того, чтоб сделать любой выбор, нужно иметь один, чётко сформулированный, как угодно сложный, критерий. А как быть, если критерий выбора неизвестен, не задан? Выбор в слепую? Случайный, равновероятный выбор. Вот универсальный и простейший критерий. Для указания в алгоритме потребности выбора одного варианта процесса, со своими данными, в языке есть встроенная функция/команда «define()». Она осуществляет выбор одной точки исполнения из существующего, на момент исполнения ёе, множества. Как это работает. Текущий процесс достигает команду «define()» и останавливается. Активным делается другой, не остановленный, процесс из списка параллельных процессов. Когда все процессы в списке стают, остановлены, это значит, что пришло время сделать выбор одного из них. Избирается одна точка исполнения из списка, а все остальные процессы закрываются. Выбранный процесс запускается на дальнейшее исполнение алгоритма. Таким образом, независимо от порядка обработки параллельных процессов, после исполнения команды «define()» объективно, гарантировано остаётся только одна точка исполнения, со своим вариантом данных. Команду определения («define()»), можно исполнить и субъективно, о чём я расскажу пожже. Вот пример использования операции автоматического выбора.

Программа на «С++»:	Программа на «Автор»:	По другому на «Автор»:

#include <stdio.h>
#include <stdlib.h>
void main(){
int i=rand()%7;
printf("i=%d\n",i);
}
	void main(){
	int i=rand()%7;
	trace("i="+i);
	}
		void main(){
		int i=rozpad({0,1,2,3,4,5,6});
		define();
		trace("i="+i);
		}

Здесь следует ввести понятия «одновариантный алгоритм» и «многовариантный алгоритм».

Одновариантный алгоритм это такой алгоритм, в процессе исполнения которого не возникает деление точки исполнения. Все алгоритмы, которые вы знали раньше – одновариантные.

Многовариантный алгоритм это такой алгоритм, в процессе исполнения которого возникает деление точки исполнения.

Вот ещё один пример многовариантного алгоритма с использованием автоматического выбора.

Программа на «Автор»:	Результат работы на экране:

int main(){
trace("Известные приветствия:");
string str = rozpad({"Привет.",
"Здравствуйте.", "Доброго времени суток."});
trace(str);
define();
trace("Я выбрал: "+str);
//getstring();
}
	Известные приветствия:
	Привет.
	Здравствуйте.
	Доброго времени суток.
	Я выбрал: Привет.

Напомню, что при выводе данных на экран из параллельных процессов, они сливаются в произвольном порядке. Таким образом, если необходимо свести точки исполнения к одной используется команда определения («define()»).

Теперь объединим полученные знания, о языке для понимания работы следующей программы:

Текст (код) программы:	Результат работы на экране:

void main(){
int i=rozpad({-3,-2,-1,0,1,2,3}); // строчка 1
if(i>0 && i%2)OFF;
if(i==0)i=100;
if(i<0)define();
trace(i);
}
	100
	2
	-3

В первой строчке кода происходит распад точки исполнения на семь независимых. Во второй строчке кода происходит условное деление вариантов на две группы. Каждый вариант значения переменной «i», для которой её значение больше нуля и не кратно двум, закрывается, удаляется. Таким образом, после исполнения второй строчки кода, гарантировано, что переменная «i» содержит значения либо отрицательных чисел, либо чисел, которые кратные двум и больше нуля. В третий строчке кода

происходит замена значения переменной «i» для каждого варианта, в котором оно раньше было равно нулю, на сто. В четвёртой строчке, происходит условное деление точек исполнения на две группы. Условная группа вариантов процессов, для которых значение переменной «i» отрицательно, проходит через команду определение одного варианта. Тут дело не в какой-то группе вариантов, а дело в том, что каждая точка исполнения кода, которая дойдёт и зайдёт в данное разветвление, в дальнейшем проходит через команду определения. Другая же условная группа не проходит через команду определения одного варианта. Таким образом, после исполнение этой строчки, гарантировано, что среди всех вариантов точек исполнения, будет только одна, для которой значение «i» отрицательное. Поймите правильно, вариантов будет множество, но только один из них с отрицательным «i». Таким образом, на экран выводится то, что вы видите.

Для того, чтобы успеть рассмотреть результат роботы программы на экране, перед тем, как она закроется, можно использовать функцию «getstring()». Она возвращает строку текста, введенную с клавиатуры в консоли. Для продолжения работы программы достаточно нажать enter. Создана в языке исключительно для отладки. Но обратите внимание. Для каждого параллельного процесса, что исполняет данную функцию, нужно снова вводить строку. По этому желательно сначала определить один рабочий процесс.

Следующая программа определяет из заданного множества чисел одно отрицательное и одно положительное число.

Текст программы:	Результат работы на экране:

void main(){
int i=rozpad({-3,-2,-1,0,1,2,3}); // строчка 1
if(i<0)define(); else define();
trace(i);
define(-1);
getstring();
}
	-2
	0
	При другом запуске:
	-3
	1

В первой строчке кода происходит деление на варианты. Во второй – деление на две группы и определение одного варианта для каждой из условных групп. Далее выводим на экран результат и ждем нажатия enter.

Функция «define()» может принимать одно числовое значение. Оно определяет приоритет определений. Первым произойдёт то определение, которое получит большее число. По умолчанию берётся ноль. Если порядок определения неважен, можно использовать одинаковое число. Вот пример кода, для которого порядок определения важен.

Текст программы А:	Результат А:	Текст программы Б:	Результат Б:

void main(){
int n=rozpad({-2,-1,0,1,2,3});
if(n>=0)define(1);
trace(n);
define(2);
trace("OK");
}
	-2
	-1
	OK
	2
	OK
		void main(){
		int n=rozpad({-2,-1,0,1,2,3});
		if(n>=0)define(1);
		trace(n);
		define(0);
		trace("OK");
		}
			-2
			-1
			0
			OK

Чтобы понять результаты работ представленных программ, нужно знать, что параллельные процессы обрабатываются по очереди. Настаёт момент, когда каждый процесс достигнет своего «define». На это время первые два числа уже будут на экране, потому что два процесса пройдут чрез «trace(n)». Затем виртуальная машина интерпретатора должна выбрать какой «define» исполнять. Для программы «А» будет избран «define(2)». Будет случайно избран один процесс из двух с отрицательным значением переменной «n». Дальше этот избранный процесс проходит через «trace("OK")» и закрывается по достижению конца программы. Остальные процессы остаются остановленные и ждут определения для «define(1)». На этот момент существует только один определитель, и он активируется. Будет случайно избран один процесс из четырёх с неотрицательным значением переменной «n». Избранный процесс далее проходит через «trace(n)» и достигает «define(2)». На этот момент существует только один процесс, он и избирается при определении. Далее исполняется «trace("OK")» и завершается процесс. Вот почему «OK» вывелось на экран два раза. Для программы «Б», первым делом исполнится «define(1)». Определяется один процесс из процессов достигших его. Далее выводится значение «n» и текущий процесс приходит к «define(2)». Теперь будет избран один из трёх процессов. Избранный процесс проходит через «trace("OK")» и закрывается. Вот почему «OK» вывелось один раза. Вот почему порядок определений может быть важен.

Язык имеет средства измерения времени работы программы. Для этого в него встроена функция «getTime()», которая возвращает количество секунд от начала компьютерной эры. Также имеется функция «timeFormat()», которая преобразует количество секунд в строку с форматом, указывающим часы, минуты и секунды.

Пример замера времени:	Результат:

void main(){
int starttime=getTime();
rozpad({1,2,3});
Sleep(63000);
define();
trace(timeFormat(getTime()-starttime));
}
	3m 9s
	
	Три минуты, девять секунд.

Сначала программа замеряла текущее время. Потом исполнение разделилось на три процесса. Исполняя каждый из них, компьютер поспал 63 секунды. После определения одного процесса выводится на экран разница во времени.

Язык позволяет задавать числовой промежуток (интервал): «x=[0;10)». Объект интервал имеет методы «getN()» и «getZ()». Так «[0;5).getN()» вернёт множество целых чисел от нуля включительно до пяти исключительно – {0,1,2,3,4}.

Текст программы А:	Результат А:

void main(){
trace([-5;10));
trace([-2;4).getN());
trace([-2;4).getZ());
}
	[-5;10)
	vector[0,1,2,3]
	vector[-2,-1,0,1,2,3]

При каждом делении текущего процесса на варианты, они помещаются в очередь. При закрытии текущего (активного) процесса, что происходит при достижении его конца программы или при исполнении команды «OFF», из начала списка, в порядке очереди выбирается следующий, не остановленный, процесс. Но порядок выборки из списка можно изменить функцией «setPrioritetProcess()». Функция принимает строку текста и реагирует на значения: "begin" – «выбирать из начала списка процессов», "end" – «выбирать из конца списка» и "random" – «выбирать случайный процесс в списке». Теперь вы имеете достаточно знаний, чтоб преступить к загадке Эйнштейна.

Загадка Эйнштейна.

1. Есть 5 домов каждый разного цвета.
2. В каждом доме живет по одному человеку отличной друг от друга национальности.
3. Каждый жилец пьет только один определенный напиток, курит определенную марку сигарет и держит определенное животное.
4. Никто из 5 человек не пьет одинаковые с другими напитки, не курит одинаковые сигареты и не держит одинаковое животное.

Вопрос: кому принадлежит рыба?

Англичанин живет в красном доме

Швед держит собаку

Датчанин пьет чай

Зеленый дом стоит слева от белого (считайте, что эти дома стоят рядом - иначе в задаче получаются не одно решение)

Жилец зеленого дома пьет кофе

Человек, который курит Pall Mall, держит птицу

Жилец из среднего дома пьет молоко

Жилец из желтого дома курит Dunhill

Норвежец живет в первом доме

Курильщик Marlboro живет около того, кто держит кошку

Человек, который содержит лошадь, живет около того, кто курит Dunhill

Курильщик сигарет Winfield пьет пиво

Норвежец живет около голубого дома

Немец курит Rothmans

Курильщик Marlboro живет по соседству с человеком, который пьет воду

Комментарий к условию:

Для начала разгадайте эту загадку сами. Фраза «Зеленый дом стоит слева от белого» совсем не означает, что эти дома находятся рядом. Но среди вариантов расстановки домов, определённых данным утверждением, найдутся и такие. По этому берём за основу условие в той степени абстрактности, в которой оно дано, без информации в скобках. Алгоритм решения данной загадки должен выдать все возможные варианты решения её. Функция «echo()» аналогична функции «trace()», только результат выводится не на экран, а в файл «out.html». Функция «reserve(5)» вернёт массив из пяти чисел, равных нулю.

```
// Решение Загадки Энштейна.
// Enshtein.txt

void main(){
	var starttime=getTime();
	setPrioritetProcess("end"); // строчка 2
	//0-цвет дома, 1-национальность, 2-напиток, 3-сигареты, 4-животное.
	var m=reserve(5);
	for(int i=0;i<5;++i)m[i]=reserve(5);
	i=rozpad( [0;5).getN() );
	m[1][i]="Англичанин";
	m[0][i]="красный";
	i=rozpad( [0;5).getN() );
	if(m[1][i]!=0)OFF;
	m[1][i]="Швед";
	m[4][i]="собака";
	i=rozpad( [0;5).getN() );
	if(m[1][i]!=0)OFF;
	m[1][i]="Датчанин";
	m[2][i]="чай";
	//Зеленый дом стоит слева от белого ([Зеленый] < [белого])
	i=rozpad( [0;5).getN() );
	j=rozpad( [0;5).getN() );
	if(i==j)OFF;
	if(m[0][i]!=0 || m[0][ j]!=0)OFF;
	m[0][i]="зелёный";
	m[0][ j]="белый";
	if(j<i)OFF;
	//if( j-i>1)OFF;// Дома стоят рядом
	if(m[2][i]!=0)OFF;
	m[2][i]="кофе";
	i=rozpad( [0;5).getN() );
	if(m[4][i]!=0)OFF;
	m[3][i]="Pall Mall";
	m[4][i]="птица";
	i=2;
	if(m[2][i]!=0)OFF;
	m[2][i]="молоко";
	i=rozpad( [0;5).getN() );
	if(m[0][i]!=0 || m[3][i]!=0)OFF;
	m[0][i]="желтый";
	m[3][p=i]="Dunhill";
	if(m[1][0]!=0)OFF;
	m[1][0]="Норвежец";
	//Курильщик Marlboro живет около того, кто держит кошку
	i=rozpad( [0;5).getN() );
	j=rozpad( [0;5).getN() );
	if(!(i-j==1 || j-i==1))OFF;
	if(m[3][i]!=0 || m[4][ j]!=0)OFF;
	m[3][s=i]="Marlboro";
	m[4][ j]="кошка";
	//Человек, который содержит лошадь, живет около того, кто курит Dunhill
	j=rozpad( [0;5).getN() );
	if(!(p-j==1 || j-p==1))OFF;
	if(m[4][ j]!=0)OFF;
	m[4][ j]="лошадь";
	i=rozpad( [0;5).getN() );
	if(m[3][i]!=0 || m[2][i]!=0)OFF;
	m[3][i]="Winfield";
	m[2][i]="пиво";
	//Норвежец живет около голубого дома
	if(m[0][1]!=0)OFF;
	m[0][1]="голубой";
	i=rozpad( [0;5).getN() );
	if(m[1][i]!=0 || m[3][i]!=0)OFF;
	m[1][i]="Немец";
	m[3][i]="Rothmans";
	//Курильщик Marlboro живет по соседству с человеком, который пьет воду
	i=rozpad( [0;5).getN() );
	if(!(s-i==1 || i-s==1))OFF;
	if(m[2][i]!=0)OFF;
	m[2][i]="вода";
	//Вопрос: кому принадлежит рыба?
	i=rozpad( [0;5).getN() );
	if(m[4][i]!=0)OFF;
	m[4][i]="рыба";
	trace(m);
	echo("Рыбка принадлежит: "+m[1][i]+"<br/>");
	echo(m);
	echo("<br/><br/>");
	define();
	echo("Времени ушло:" + timeFormat(getTime()-starttime));
}
```

Исходник программы лежит в пакете с языком программирования в папке «author\code\example\Enshtein.txt».

Сам интерпретатор «Автор» с множеством примеров можно скачать: http://monstr.domivo4ka.com в разделе «искусственный интеллект».

Результат работы программы – текст в файле «out.html»:

```
Рыбка принадлежит: Немец
vector[
vector["зелёный","голубой","белый","желтый","красный"],
vector["Норвежец","Немец","Швед","Датчанин","Англичанин"],
vector["кофе","вода","молоко","чай","пиво"],
vector["Pall Mall","Rothmans","Marlboro","Dunhill","Winfield"],
vector["птица","рыба","собака","кошка","лошадь"]]

Рыбка принадлежит: Датчанин
vector[
vector["зелёный","голубой","белый","желтый","красный"],
vector["Норвежец","Немец","Швед","Датчанин","Англичанин"],
vector["кофе","вода","молоко","чай","пиво"],
vector["Pall Mall","Rothmans","Marlboro","Dunhill","Winfield"],
vector["птица","кошка","собака","рыба","лошадь"]]

Рыбка принадлежит: Норвежец
vector[
vector["зелёный","голубой","желтый","красный","белый"],
vector["Норвежец","Немец","Швед","Англичанин","Датчанин"],
vector["кофе","вода","молоко","пиво","чай"],
vector["Marlboro","Rothmans","Dunhill","Winfield","Pall Mall"],
vector["рыба","кошка","собака","лошадь","птица"]]

Рыбка принадлежит: Датчанин
vector[
vector["зелёный","голубой","белый","красный","желтый"],
vector["Норвежец","Немец","Швед","Англичанин","Датчанин"],
vector["кофе","вода","молоко","пиво","чай"],
vector["Pall Mall","Rothmans","Marlboro","Winfield","Dunhill"],
vector["птица","кошка","собака","лошадь","рыба"]]
```

```
Рыбка принадлежит: Немец
vector[
vector["зелёный","голубой","красный","желтый","белый"],
vector["Норвежец","Немец","Англичанин","Датчанин","Швед"],
vector["кофе","вода","молоко","чай","пиво"],
vector["Pall Mall","Rothmans","Marlboro","Dunhill","Winfield"],
vector["птица","рыба","лошадь","кошка","собака"]]

Рыбка принадлежит: Датчанин
vector[
vector["зелёный","голубой","красный","желтый","белый"],
vector["Норвежец","Немец","Англичанин","Датчанин","Швед"],
vector["кофе","вода","молоко","чай","пиво"],
vector["Pall Mall","Rothmans","Marlboro","Dunhill","Winfield"],
vector["птица","кошка","лошадь","рыба","собака"]]

Рыбка принадлежит: Немец
vector[
vector["желтый","голубой","красный","зелёный","белый"],
vector["Норвежец","Датчанин","Англичанин","Немец","Швед"],
vector["вода","чай","молоко","кофе","пиво"],
vector["Dunhill","Marlboro","Pall Mall","Rothmans","Winfield"],
vector["кошка","лошадь","птица","рыба","собака"]]

Времени ушло:24s
```

Если закомментировать вторую строку программы – выборка будит происходить с начала списка. Результатом будут те же варианты ответа, но «Времени ушло:34m 0s».

А если раскомментировать условие в строке с меткой «Дома стоят рядом» получим следующий результат:

```
Рыбка принадлежит: Немец
vector[
vector["желтый","голубой","красный","зелёный","белый"],
vector["Норвежец","Датчанин","Англичанин","Немец","Швед"],
vector["вода","чай","молоко","кофе","пиво"],
vector["Dunhill", "Marlboro","Pall Mall","Rothmans","Winfield"],
vector["кошка","лошадь","птица","рыба","собака"]]

Времени ушло:7s
```

Из текста программы видно, что алгоритм решения взят из самого условия задачи, без дополнительных размышлений над ним. Никакой оптимизации я не проводил. Таким образом, язык «Автор», во многом, даёт возможность постановки задачи для машинной обработки, непосредственно из условия задачи. Что упрощает способ перевода задачи в алгоритм решения её.

Язык содержит указатели. Способ их использования ни чем не отличается от «С», но природа их другая. Любой указатель это просто строка текста, которая содержит данные про область и имя переменной в карте памяти. Этих данных достаточно, для идентификации переменной, на которую ссылается указатель. Таким образом, даже при необходимости деления переменной на варианты соответствующие параллельным процессам, указатель на неё останется верный своему варианту.

Язык не имеет строгой типизации переменных. Тип переменных воспринимается не более чем комментарий для наглядности. Соответственно при инициализации переменных никакого, неявного, приведения типов не происходит. Более того объявлять переменные необязательно.

Вот пример многовариантной программы с использованием указателя.

Текст программы:	Результат:

void main(){
int n;
var p=&n;
n=rozpad({5,9});
++n;
trace("*p=="+*p);
define();
getstring();
}
	*p==6
	*p==10

Язык содержит тип указатель на команду в алгоритме. Этот тип можно преобразовать в строку и получить текст команды, на которую он указывает.

Текст программы:	Результат на экране:

void main(){
pos=getFunctionRoot(getThisFunctionName());
pos2=pos;
trace(++pos);
trace(--pos2);
getstring();
}
	getFunctionRoot(getThisFunctionName())
	getstring()

Функция «getThisFunctionName()» возвращает имя текущей, исполняемой функции. Функция «getFunctionRoot» возвращает первый и последний (он один) узел указанного алгоритма.

Теперь рассмотрим следующую задачу. Найти расстановку ферзей на шахматной доске так, чтоб они не угрожали друг другу.

В частности решим такие задачи:

1) Найти одну, случайную, комбинацию ферзей на шахматной доске заданного размера, например 20*20, так, чтоб они не угрожали друг другу, и на доске не оставалось клеток, находящихся не под боем.

2) Найти все возможные комбинации ферзей на шахматной доске, размером 7*7, так, чтобы они не угрожали друг другу, и на доске не оставалось клеток не под боем. Но только одну из найденных комбинаций вывести на экран.

3) Найти, и вывести на экран, все возможные комбинации, наибольшего количества ферзей на шахматной доске, размером 4*4, так, чтоб они не угрожали друг другу.

Обратите внимание на задачи 2 и 3. Сопоставьте их. Представьте, что их нужно задать для общего случая, с размерами доски n*n. Это одна и та же задача, за исключением того, что в одной не задан критерий выбора, а в другой задан. В одной следует сделать случайный выбор, а в другом следует выбрать те комбинации ферзей, в которых наибольшее количество ферзей. В одной нужно исполнить «define» объективно, в другом субъективно. Субъективное определение это выбор альтернатив по конкретному, известному, критерию. Но, обратите внимание, что нельзя просто заменить «define» каким-то кодом, поскольку нужно сопоставлять данные из разных процессов. Любой код будет работать со своим вариантом значений переменных и доступа к параллельным данным не имеет.

Из этого следует ещё одно техническое решение. Весь многовариантный алгоритм, со всеми своими вариантами, находится внутри одного программного слоя, а обслуживающий его одновариантный алгоритм исполняется в другом. Обслуживающий алгоритм можно назвать также интерпретатором второго порядка. Он имеет доступ ко всем параллельным процессам подопытного слоя. Он какбы следит за ходом исполнения подопытного алгоритма и в момент, когда дело доходит до выбора, берёт ответственность на себя.

Вот пример субъективного выбора одного из процессов.

Текст программы:	Результат:

void f1(){
x=rozpad({9,4,67,3,28});
define();
trace(x);
}
 	
void main(){
var pos=getFunctionRoot("f1");
++pos;
int n=createNewLaver();
int p=createNewProcess(n,pos);
runLaver(n);
//nVariantov=getVariantov(n);
m=getValuesInLaver(n,"x");
int maxi=0;
for(i=1;i<m.size();++i)if(m[maxi]<m[i])maxi=i;
for(i=0;i<m.size();++i)if(maxi!=i)LaverOFF(n,i);
runLaver(n);
deleteLaver(n);
getstring();
}
	67

Функция «createNewLaver()» создаёт новый программный слой и возвращает номер его. Функция «createNewProcess()» создаёт новый процесс в программном слое с указанным номером и узлом алгоритма (точкой исполнения). Возвращает номер созданного процесса в списке всех процессов указанного программного слоя. Функция «runLaver()» передаёт активность другому программному слою с указанным номером. Далее управление переходит процессу, исполняющему функцию «f1()». Происходит деление на варианты. Все процессы данного программного слоя, по очереди, достигают команды «define()» и останавливаются. По остановке последнего процесса, виртуальная машина передаёт управление родительскому программному слою. Функция «getVariantov()» возвращает количество процессов в указанном программном слое. Функция «getValuesInLaver()» возвращает массив значений переменной с указанным именем, собранный из всех параллельных процессов в указанном программном слое. Порядковый номер значения в массиве совпадает с номером процесса, из которого были взяты данные. Далее находим номер ячейки в массиве с максимальным значением. Функция «LaverOFF()» подготавливает процесс с указанным номером, в указанном программном слое к закрытию. Само закрытие процесса произойдёт сразу после активации его программного слоя. Таким образом, закрываются все процессы, кроме того, в котором значение переменной «x» наибольшее. По завершению работы последнего процесса в подопытном программном слое, управление передаётся родительскому программному слою. Далее следует удалить подопытный слой функцией «deleteLaver()».

Теперь вернёмся к предыдущей задачи. Вот текст программы для решения её на языке «Автор».

```
// author\code\example\ferzi.txt
// Программа находит расстановку ферзей на шахматной доске так,
// чтоб они не угрожали друг другу.
//-------------------------------------------------------------------------------------------
void main(){
        randomFishka(20);
        trace("----------------------------------------");
        procces(7); // n*n - размер поля
        trace("----------------------------------------");
        interpretator(4);
}

//интерпретатор второго порядка для функции procces(n)
void interpretator(var nn){
        var pos=getFunctionRoot("procces");
        ++pos;
        int n=createNewLaver();
        int p=createNewProcess(n,pos);
        var names=pos.getNamesStart();
        setValueInLaver(n,p,names[0],nn);
        runLaver(n);
        var NP=getVariantov(n);
        trace("Variantov:"+NP);
        var ks=getValuesInLaver(n,"k");
        trace(ks.export());
        int max=0;
        for(int i=0;i<ks.size();++i)if(max<ks[i])max=ks[i];
        for(i=0;i<ks.size();++i)if(max>ks[i])LaverOFF(n,i);
        runLaver(n);
        NP=getVariantov(n);
        for(i=1;i<NP;++i)LaverOFF(n,i);
        runLaver(n);
        deleteLaver(n);
}

//найдет все решения задачи ферзей.
void procces(var n){
        var m=initP(n),x,y=0,ok,k=0;
        while(!nozero(&m)){
                do{
                        ok=0;
                        for(int i=0;i<n;++i)if(m[i][y]==0){ok=1;break;}
                        if(!ok)++y;
                        if(y>=n)OFF;
                        }while(!ok);
                x=rozpad([0;n).getN());
                if(m[x][y])OFF;
                protect(x,y,&m);
                ++k;
                //trace(toString(&m));
                }
        define();
        trace("ferzei "+k+":");
        trace(toString(&m));
        define();
        trace("Pres enter for exit.");
        getstring();
}
```

```
//найдет одно случайное решение.
void randomFishka(var n){
	var m=initP(n),x,y;
	while(!nozero(&m)){
		x=rozpad([0;n).getN());
		y=rozpad([0;n).getN());
		if(m[x][y])OFF;
		define();
		protect(x,y,&m);
		}
	trace(toString(&m));
	getstring();
}

//---------------------------------------------------------------------------------------
//создаст матрицу - поле игры.
void initP(int n){
	var m=reserve(n);
	for(int i=0;i<n;++i)m[i]=reserve(n);
	return m;
}

//вернёт истену если нет клеток не под боём.
bool nozero(var*m){
	int n=m->size();
	for(int x=0;x<n;++x)
	for(int y=0;y<n;++y)if((*m)[x][y]==0)return 0;
	return 1;
}
```

```
//покроет ячейки которым угрожает фигура.
void protect(int x,int y,var*m){
	int n=m->size(),p;
	for(p=0;p<n;++p){
		(*m)[x][p]=1;
		(*m)[p][y]=1;
		}
	for(p=0;p<n;++p){
		if(x+p<n && y+p<n)(*m)[x+p][y+p]=1;
		if(x-p>=0 && y-p>=0)(*m)[x-p][y-p]=1;
		if(x-p>=0 && y+p<n)(*m)[x-p][y+p]=1;
		if(x+p<n && y-p>=0)(*m)[x+p][y-p]=1;
		}
	(*m)[x][y]=2;
}


//превратит массив в строку текста для вывода на екран.
string toString(var*m){
	string s="";
	int n=m->size(),x,y;
	for(int y=0;y<n;++y){
	for(int x=0;x<n;++x){
		if((*m)[x][y]==0)s+=".";
		if((*m)[x][y]==1)s+="O";
		if((*m)[x][y]==2)s+="F";
		}
	s+="\n";
	}
	return s;
}
```

Результат роботы программы на экране.

```
OOOOOOOOOOOOOOFOOOO
OOOOOOOOOOOOFOOOOOOO
OOOOOOOOOOFOOOOOOOOO
FOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOFOOOOOO
OOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOF
OOOFOOOOOOOOOOOOOOOO
OOOOOOOOFOOOOOOOOOOO
OOOOOOOOOOOOFOOOOOOOO
OOOOFOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOFOO
OOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOFOOO
OOOOOFOOOOOOOOOOOOOO
OOOOOOOFOOOOOOOOOOOO
OOFOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOFO
OOOOOOFOOOOOOOOOOOOO
OOOOOOOOOOOOOOFOOOOO
```

```
----------------------------------------
ferzei 6:
OFOOOOO
OOOFOOO
OOOOOOF
FOOOOOO
OOFOOOO
OOOOOFO
OOOOOOO

Pres enter for exit.
```

```
----------------------------------------
Variantov:6
{3,3,3,3,4,4}
ferzei 4:
OFOO
OOOF
FOOO
OOFO

ferzei 4:
OOFO
FOOO
OOOF
OFOO

Pres enter for exit.
```

Метод указателя на узел алгоритма «pos.getNamesStart()» возвращает массив имен переменных объявленных в функции - владельца данного алгоритма.

Программа состоит из трёх основных функций: «randomFishka», «procces» и «interpretator». randomFishka() – быстро расставит случайным образом ферзей на доске и выдаст одно из возможных решений. procces() – найдёт все возможные решения задачи расстановки ферзей. interpretator() – выберет из всех решений те, у которых максимальное количество ферзей.

Язык программирования «Автор» призван дополнить возможности C++ а не заменить его. По этому в мой язык встроена возможность подключения модулей написанных на C++ в качестве dll библиотек.

Функции:	Описание:
LoadDLL(namedll);	Загружает dll с указанным именем.
SendDLL(namedll,"string");	Передаёт данные в указанный модуль.
RecvDLL(namedll)	Возвращает данные из указанного модуля.
unLoadDLL(namedll);	Выгружает указанный dll модуль.

Глобальная переменная «WAY» хранит путь к папке с запускаемым файлом интерпретатора.

Теперь, для демонстрации, можно подключить один из модулей в пакете с языком. Например, модуль обработки простых математических вопросов «what.dll». Вот программа, которая выводит значение переменной «y» из уравнения «y*y+7*y-9=x-y».

Текст программы:	Результат работы на экране:

void main(){
var namedll="what.dll",what="";
LoadDLL(namedll);
SendDLL(namedll,"WAY:"+WAY);
//Текст вопроса
what="?: get equality for (y*y+7*y-9=x-y) unknown {y}";
SendDLL(namedll,what);
n=RecvDLL(namedll);
if(n==0 || n=="")trace("no rejoin.");
SendDLL(namedll,"Rejoin");
for(i=0;i<n;++i)trace(RecvDLL(namedll));
unLoadDLL(namedll);
}
	y={pow(25+x,0.5)-4, -4-pow(25+x,0.5)}

Полный текст программы теста DLL модуля, который способен отвечать на некоторые математические вопросы, вы найдете в «author\code\example\questions_test.txt».

Простейшая само изменяющаяся программа.

Следующая программа подсчитывает количество собственных запусков. В исходном коде программы, в первой строчке объявлена переменная – счётчик запусков. Дальше идёт код трансформации этой переменной.

Текст программы:	Результат работы на экране:

void main(){
i=0;
pos=getFunctionRoot(getThisFunctionName());
++pos;
++i;
trace("This function called: "+i);
replaceFormul(pos,"i="+i);
}
	This function called: 1
	Результат работы на экране через 10 запусков:
	This function called: 10

Простейшая самообучающаяся программа поиска простых чисел.

Текст исходного кода программы можно найти в файле: «author\code\example\prosto.txt».

В первой строке кода объявлен массив с первыми числами ряда простых чисел: «m={2,3,5,7,11,13}». Программа запрашивает ввести число из клавиатуры и сообщает, является ли оно простым числом. Вот только в массив приходится добавлять продолжение ряда, чтоб дать ответ для более больших чисел. Но каждый рас просчитывать нерационально. Можно трансформировать стартовый массив поиска, что и делает программа. Таким образом, после анализа число «100», массив изменится и станет таким: «m={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97}». Это можно увидеть в дублирующем файле «author\code\example\prosto.code».

Напомню, что сам интерпретатор «Автор» с множеством примеров можно скачать: http://monstr.domivo4ka.com в разделе «искусственный интеллект».