Цель:

Создать одностраничный сайт-тренажёр, который позволял бы «натаскивать» студентов на понимание того, в каком порядке проверяются условия и выполняются ветки в развилках (if-elseif-else), когда заканчивается цикл и когда нужно его продолжать, чем отличается цикл с постусловием, цикл с условием завершения вместо условия продолжения и т.п.

Контекст:

В существующей модели присутствуют только действия, управляющие структуры и их условные выражения. Не используются понятия переменной, операции (вроде + или \*), нет классов и областей видимости. Понятие «значение» фигурирует только постольку, поскольку нужно указать булевый результат «как бы вычисленного» условия цикла или альтернативы.

Идея:

Мы все понимаем алгоритмические структуры в разной степени, но качество этого понимания трудно проверить непосредственно (можно косвенно, но это имеет серьёзные минусы). Для этого вводится понятие **трассы** – материализованное представление о порядке действий, совершаемых программой при исполнении алгоритма.

Проще всего представить себе трассу как выписанные в столбик на бумажку все действия (пользовательского уровня), которые мы прошли под отладчиком в режиме пошаговой трассировки. Сюда войдут как строки типа «выполнился printf», так и строки типа «началась функция main» и «закончилась функция main». Каждую строку будем называть актом, началом акта или концом акта.

Глядя на все акты трассы, можно говорить о том, как именно выполняется программа (и кто как это понимает), без двусмысленностей.

Практика:

С помощью тренажёра студент будет решать задачи. Пока существует такая постановка задачи:

Постройте правильную трассу   
по заданному алгоритму   
и значениям управляющих условий

Причём, может предлагаться построить трассу «с нуля», или достроить уже начатую.

Задача:

Спроектировать и реализовать клиентскую часть тренажёра с простейшей авторизацией (иметь виду возможное расширение до LTI).

В перспективе – с интерфейсом для выбора заданий, заранее подготовленных преподавателем (первоначально условия задач могут рассылаться хоть по почте).

Тренажёр **должен** обеспечивать:

* ввод условий задачи, их валидацию,
* процесс решения задачи в виде составления трассы из актов и редактирования трассы (перемещая/удаляя/добавляя акты),
* отправку созданной трассы на сервер (возможно, понадобится также идентифицировать студента в запросе) и получение вердикта с ошибками для некоторых актов,
* отображение ошибок студенту,
* завершение задачи после успешного построения трассы до конца (включение запрета на редактирование правильной трассы).

Следует сразу заложить возможность интернационализации (рус./англ.). Синтаксис алгоритмов и трасс уже поддерживает два языка (в основном).

Дальнейшие подробности – в детальном описании ниже.

На веб-странице тренажёра **должно** быть:

* Алгоритм:
  + состоит из: простых действий, управляющих структур (следования, альтернативы, циклы) и условий альтернатив и циклов, значения которых направляют ход выполнения программы,
    - каждый из элементов алгоритма имеет внутренний ID (уникальный в пределах одного алгоритма) для ссылок из трассы, и номер строки для ссылок на строку в GUI;
  + является частью условия задачи (устанавливается преподавателем) и не меняется на протяжении работы студента над задачей,
  + априори не содержит ошибок (всегда верен),
  + желательно, **должен** иметь цветную раскраску (подсветку синтаксиса наподобие уже реализованной),
  + на каждом действии, которое может, так или иначе, появиться в трассе, **нужно** сделать активную зону (кнопку, появляющуюся при наведении или т.п.), при нажатии на которую в трассу **должен** добавляться ещё один правильно отформатированный акт:
    - учесть создание разных актов:
      * *выполнить* – для простых действий,
      * *начать* и *закончить* – для составных,
      * *начать/закончить итерацию* – для циклов,
      * *начать/закончить ветку* – для альтернатив.
* Управляющие условия альтернатив и циклов:
  + цепочка величин ***Истина*** / ***Ложь*** для каждого условия в алгоритме (задающая значения, принимаемые этим условием при каждой очередной проверке по ходу выполнения алгоритма),
  + также являются частью условия задачи и не меняются,
  + задают конкретную трассу (коих может быть множество) для данного алгоритма,
  + **должны** отображаться по одному (где-то напротив своих условий в алгоритме), по мере добавления студентом в трассу актов соответствующего условия (чтобы не сбивать его «прошлыми» и «будущими» значениями, не актуальными в текущий момент).
* Трасса:
  + Трасса – это запись пошаговой трассировки алгоритма, включающая все простые действия, а также начала и концы всех составных действий, т.е. список актов, порядок которых и является предметом рассмотрения, поиска ошибок и корректировки понимания студента,
  + желательно, **должна** иметь цветную раскраску (подсветку синтаксиса наподобие уже реализованной),
  + создаётся студентом в качестве решения задачи:
    - для этого «из алгоритма» **должны** создаваться акты – кликами или «вытягиванием» drag&drop – из актов формируется трасса;
    - **можно** спроектировать GUI трассы в виде *(согласовать!)*:
      * модифицируемого списка (свободные акты недопустимы, строки актов можно переставлять, удалять, вставлять снова из алгоритма);
      * или в виде рабочей области, где плитки актов можно создавать, таскать и соединять в один список трассы (наподобие Scratch);
  + может содержать ошибки (допущенные студентом),
  + может быть полной и неполной (не доведённой до последнего акта в программе),
  + но **не может** содержать «приемлемых» пропусков в середине и в начале (это уже будут ошибки),
  + **не должна** ограничивать фантазию студента насчёт порядка актов в трассе (это мы и хотим исправить в его понимании, давая объяснение на каждую строку с неправильным актом),
  + но **не должна** допускать появление «внутреннне невалидных» актов:
    - каждый акт содержит:
      * фазу выполнения: **выполнилось** (атомарный акт занимает одну строку трассы) и **началось** / **закончилось** (две строки трассы на начало и конец составного акта),
      * указание на выполняемое действие/условие по имени и по ID (внутренне),
      * номер (раз) исполнения в трассе: *1-й*, *2-й* и т.д.,
      * собственный ID,
      * номер строки (в GUI),
      * только для актов условных выражений: метку со значением условия в этот раз;
    - подобные акты (отличающиеся только разом исполнения), расположенные по трассе по порядку, **должны** иметь номер исполнения как в натуральном ряду (1, 2, 3, 4, …): это означает, что операции по модификации трассы (перемещение / удаление / вставка актов) не должны нарушать этот порядок, т.е. он должен поддерживаться автоматически;
    - то же касается и назначения значений актам управляющих условий по порядку, указанному при алгоритме;
    - в случае попытки создать акт выражения, для которого уже не существует заданного значения (задано N значений, а студент пытается поставить акт N+1, N+2, …), внешне вести себя так же (не давать подсказку о том, что значения закончились) и использовать значение по умолчанию:
      * обычно – **false** (чтобы максимально быстро закончить все повторы);
      * для условия цикла do-until – **true** (т.к. true прерывает его);
  + GUI **должно** позволять выдавать сообщения об ошибках в трассе, привязанным к отдельным строкам трассы,
  + проверка трассы осуществляется на стороне сервера, для чего её вместе с алгоритмом **нужно** отсылать на проверку (как делается сейчас на [vds84.server-1.biz:2020/iswc/demo](http://vds84.server-1.biz:2020/iswc/demo))
  + **нужно** предусмотреть возможность отсылать не только готовую трассу, но и частичную – в процессе её формирования студентом:
    - учесть задержку сервера – расчёт ошибок может длиться 5-15 сек;
* Кнопка «Отправить на проверку»:
  + Отсылает текущее состояние трассы на проверку. Если это не сделано автоматически.
* Кнопка «Сброс/Заново/Очистить трассу»:
  + Сброс всех изменений и возврат в начало решения.
* Кнопка «Завершить решение задачи»:
  + Прервать решение или выйти из решённой задачи (перейти к следующей и т.п.)

**Пример**

К примеру прикладывается файлик «первая\_трасса.json» с примером формата алгоритма и трассы для передачи на сервер (пока это не реализовано, и они посылаются просто текстом). Пример не на 100% полон, и финальная спецификация обмена будет уточнена, но его достаточно для понимания структуры и сути данных.

Алгоритм:

* написан на псевдокоде, аналогичном по сути любому ЯП,
* с явным именованием составных действий (чтобы потом ссылаться) в комментарии справа от первой (главной) строки этой структуры (альтернатива, цикл, именованное следование)
* плюс значения управляющих условий («… **->** ***ложь***» ; «…**->** ***истина***» - здесь по одному значению на всю программу, т.к. больше не нужно, но в случае с циклами может быть и несколько значений, записанных через запятую):

*// алгоритм example\_alt (<< имя алгоритма)*

**{**

поприветствовать

**если** ответ\_позитивный **->** ***ложь*** *// по\_ответу <<- первое слово комментария - это имя сложной структуры (здесь - развилки)*

поддержать\_тон

рассказать\_анекдот

**иначе** **если** ответ\_негативный **->** ***истина***

**{**

удивиться

узнать\_что\_случилось

посочувствовать

**}**

**иначе**

спросить\_о\_настроении

пожелать\_удачи

**}**

Трасса:

* плоская последовательность актов (отступы в примере только для красоты)
* акты бывают двух видов: **выполнилось** (атомарный акт) и **началось** / **закончилось** (составной акт, имеющий другие акты внутри себя, т.е. между началом и концом)
* каждый акт ссылается на элемент алгоритма по имени (в текстовой записи) и по ID в сериализованной форме (см. пример в JSON):

*// example\_alt первая трасса*

*// в строке выше - имя трассы после имени алгоритма*

**{**

**началась** **программа**

поприветствовать **выполнилось** **1-**й раз

**началась** **развилка** по\_ответу **1-**й раз *// начало развилки*

**условие** **(**ответ\_позитивный**)** **выполнилось** **1-**й раз **-** ***ложь***

**условие** **(**ответ\_негативный**)** **выполнилось** **1-**й раз **-** ***истина***

**ветка** **условия** **(**ответ\_негативный**)** **началась** **1-**й раз

удивиться **выполнилось** **1-**й раз

узнать\_что\_случилось **выполнилось** **1-**й раз *// в трассе можно добавлять любые комментарии*

посочувствовать **выполнилось** **1-**й раз

**ветка** **условия** **(**ответ\_негативный**)** **закончилась** **1-**й раз

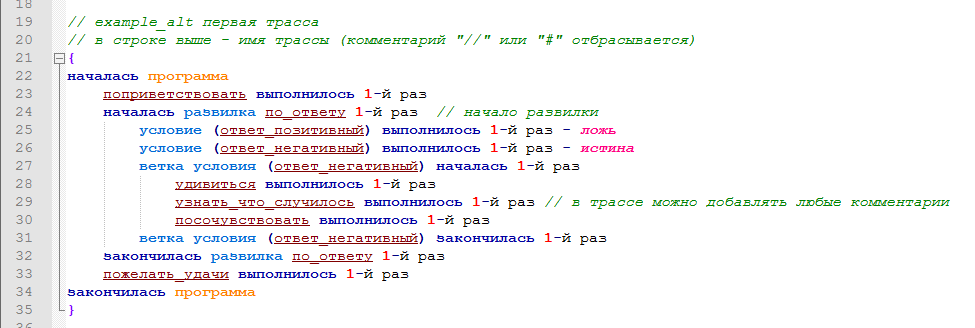
**закончилась** **развилка** по\_ответу **1-**й раз

пожелать\_удачи **выполнилось** **1-**й раз

**закончилась** **программа**

**}**

Трасса со строками – скриншот (в JSON каждый акт имеет номер строки по ключу "line\_index" – сверьте строки на картинке с примером в JSON):



P.S.

Примеры алгоритмов и трасс (где они заключены в { } ) см. в файлах:

* github.com/den1s0v/c\_owl/master/blob/handcrafted\_traces??

Для форматирования цветом у нас применяются:

* Пользовательская подсветка в Notepad++ : github.com/den1s0v/c\_owl/master/blob/trace\_gen/NPP-UDL\_highlighting/Трасса\_npp.xml??
* «mode» для web-редактора CodeMirror, построенная на тех же принципах (см. в конце скрипта <http://vds84.server-1.biz:2020/iswc/demo/static/script.js>).