Алгоритмы и их реализация в программе «Меркурий».

Д.В. Анисимов

24 июля 2004 г.

Содержание

1 Зачем я это сделал (написал).

Давно пора ядрена мать умом Россию понимать!

Меня с самого раннего детства возмущали заявления, что машинный перевод невозможен, что перевод, это такое искусство, которое по силам очень немногим особо выдающимся персонажам. И особенно меня возмущало, что люди владеющие иностранным языком, считали себя умнее прочих (например, владеющих математикой.)

Этой своей статьей я хочу, снять покров мистики с машинного перевода, рассказать программистам, как с помощью простых правил можно обьяснить компьютеру алгоритм перевода. Разумеется, я страдаю манией величия, однако я не считаю «Меркурий-Правду» идеальной программой. Я уверен, что найдутся достаточно талантливые и великодушные люди, которые захотят улучшить качество моих программ. Этот текст призван избавить моих помощников и последователей (если таковые будут) от необходимости делать работу, которую я уже сделал, совершать те открытия, которые я уже совершил. Чтобы мои последователи пошли далее меня.

«Меркурий» - это не только и не столько программа - это в основном научная работа, которая должна "пробить дырку в технологической стене", показать, что предлагаемые подходы могут эффективно использоваться при постоении программ-переводчиков. Сверхзадача состояла в том, чтобы сделать ненужной услуги коллектива программистов для построения переводчика, сделать задачу настолько легкой, чтобы давать ее одному студенту лингвистического ВУЗа в качестве курсовой работы.

Так же как Чарльз Энтони Хоар, я считаю, что хорошая программа - это хороший алгоритм + хорошие данные. По-этому в моих описаниях данным уделяется очень много внимания - 2/3 текста. Описание алгоритма выглядит примерно так «Вот из этих данных надо сделать вот те». Мне кажется при наличии открытых исходников такое описание является наилучшим компромисом между подробностью описания и обьемом текста.

2 Замечания по стилю программирования.

Я - человек со здоровым (то есть очень большим) консерватизьмом. Это проявляется во всем - во внешности, одежде, политических симпатиях, семейной жизни, ну и разумеется в исходниках.

Я пишу не на C++, я пишу на "C с классами". Это значит что я не использую многих современных возможностей языка и многих современных средств. Это мой эстетический вкус что-ли - добиваться результата используя абсолютный минимум действительно необходимых инструментов.

Я не использую:

1. темплейтов (не то, чтобы я их противник, просто так получилось).
2. производных классов (иногда таки использую, но не злоупотребляю)
3. перегрузку операторов (оператор [] - исключение)
4. операторы new delete для захвата массивов, а предпочитаю calloc и free (опять таки, я не противник, просто часто renew сделать хочется)
5. не делаю закрытия данных (это потому что пишу один и сам с собой всегда могу договориться куда лазить не надо. То есть мне не нужен специальный инструмент для того чтобы помнить, что если в классе какое-то поле обьявлено public, то это только для чтения)
6. библиотеку STL, хотя просто напрашивается использование классов STL в качестве контейнеров.

7. Lex/Yacc хотя мне понадобилось разбирать файл grammar.txt
8. Язык perl и те же самые Lex/Yacc при работе со словарями.
9. Базы данных a la SQL.
10. И уж совсем мне была не нужна Corba.

Может быть эти средства более эффективны для каждой конкретной задачи, но прикиньте, сколько всего мне не понадобилось учить, поскольку я работал одним инструментом!

Кроме аскетизма мой стиль программирования имеет еще некоторые особенности.

Единственный способ обработки ошибок, который я использую - это throw(-1) с последующим перехватом где-нибудь в самом верху.

Я люблю выстраивать данные в виде одной большой пирамиды. То есть где-то есть обьект в единственном экземпляре —"Все данные". Этот обьект — глобальная переменная, которая видна отовсюду. Этот обьект содержит в себе какие-то классы, эти классы содержат еще какие-то классы и так далее. К любым данным можно обратиться вот так AllData.a[i].b[i1].c[i2].data Сейчас так делать не модно, но... я так люблю.

Все классы и структуры начинаю с префикса «t_». А переменную этого класса обзываю так-же как и класс, только без префикса.

Люблю, когда открывающая скобка стоит над закрывающей.

Если у меня в нескольких структурах есть поле с одинаковым смыслом, я его обзываю одинаково.

Не стесняюсь дублировать одну и ту же информацию в разных данных программы, если это приводит к упрощению программы.

В качестве строк использую char* и char []. Размер строк не контролирую. Это от лени. Я потом поправлю.

Все массивы начинаю с нуля. Индекс в массив -1 означает, что значение индекса отсутствует или не вычислено.

Люблю писать коментарии по-русски (чтобы в свою же программу со словарем не лазить) Считаю, что наилучшей документацией на программу является сама программа. По этой причине больше половины этой статьи - это куски исходников, в которые добавлены дополнительные коментарии. Мне кажется, что программисту во многих случаях будет понятнее если текст написан на C++, а не на русском языке.

2.1 О едином именовании полей классов

Если у меня в нескольких структурах есть поле с одинаковым смыслом, я его обзываю одинаково. В этой подглаве я перечисляю наиболее часто встречающиеся поля классов и поясняю их смысл.

Name[] - имя чего нибудь (обычно имя конструкции, составляющей конструкции, параметра). Применяется в основном в классах внутри Grammar)

i_part - номер части речи в языке источника или приемника.

type - тип конструкции («Выбор» «Структура» и их разновидности)

Param - грамматические параметры конструкции или составляющих структуры.

Word - составляющие конструкции.

From и To - какие-нибудь два обьекта одного типа внутри одого класса, первый относится к языку-источнику а второй к языку-приемнику. Напимер, обьекта типа t_Struct внутри класса t_Trans. t_Trans - одно правило перевода, t_Struct - это как выгладит фраза на языке оригинала и языке перевода.

up, down, n_down - индексы, образующие дерево. up - индекс родителя, down- ссылка на первого потомка, n_down - число потомков. Предполагается, что все потомки всегда расположены одним куском. То есть я всегда упорядочиваю дерево, с которым работаю. Вот такой я зануда. Зато живется мне легко.

index - каким по счету потомком является эта вершина, по отношению к своему родителю; какой по счету составляющей конструкции является эта. В принципе это избыточные данные, но ооочень удобные.

i_word - с какого слова начинается конструкция.

i_last_word - с какого слова начинается следующая конструкция (то есть эта заканчивается на i_last_word-1 слове).

i_struct - номер конструкции в Grammar

Form - грамматическая форма слова или конструкции

i_slowo - Если слово или выражение найдено в словаре переводов, то i_slowo - это индекс строки, в которой оно было найдено. (На самом деле индекс в массиве Record. Разумеется, порядок расположения слов в Record не соответствует текстовому представлению словаря. Особенно, когда словарей больше одного). Если -1 - значит этого выражения в словаре нет. (Не путайте пожалуйста с i_word.)

i_slowo1 - (бывает заполнен только у данных языка-приемника) способ (вариант) перевода. Это если слово или выражение можно перевести более чем одним способом.

n_slowo - число вариантов перевода

3 Какой класс зачем.

Эта глава описывает какой класс предназначен для какой задачи. Описание дается с точки зрения "взляда снаружи". Внутреннее устройство и используемые алгоритмы описываются (если описываются) в последующих главах.

3.1 Класс t_List

Класс t_List. Это класс-контейнер, который изображает из себя массив. (Как например vector в STL). Этот класс в моих программах используется с незапамятных временю. Поскольку от STL мне был нужен только класс похожий vector, то я решил не стрелять из пушки по воробьям и не использовать STL. Тем более, что я не люблю STL.

Решение спорное, но... а какая нафиг разница?! Мне так удобнее. Если это будет создавать неудобства, - переделать недолго.

Класс t_List описан в файле list.h.

```
члены класса:
   t    *list ;                // массив
   long  j ;                   // число заполненных элементов в массиве
   long  n ;                   // число элементов, захваченных calloc-ом

функции члены:
 t1( void );           // конструктор
~t1( void );           // деструктор
   long  size( void );         // возвращает число элементов в массиве
   void  operator = ( t1 &a ); // копирует массив в другой массив
   t &   operator []( long i );// обращение к i-му элементу массива
   long  add( t &v );          // добавление одного элемента в конец массива
                               // возвращает индекс добавленного элемента
```

```
void  resize( void );       // делает Realloc на фактическую длинну массива
                            // по моему нигде не используется
void  init( void );         // обнуляет массив (исключает из него все элементы)
void  del( void );          // освобождает память от массива
```

Все эти удобства реализованы тремя макросами:

```
DECLARE_LIST_TYPE( Класс, КлассМассив )
DEFINE_LIST_TYPE ( Класс, КлассМассив )
DEFINE_LIST_BTYPE( Класс, КлассМассив )
```

Чтобы воспользоваться всеми этими удобствами нужно сделать следующее:

1) Описать прототип класса-списка

```
DECLARE_LIST_TYPE( Класс, КлассМассив )
```

"Класс" - это класс, который Вы хотите поместить в массив.
"КлассМассив" это класс, содержащий массив обьектов типа "Класс".

2) Завести переменную типа "КлассМассив".

3) Где-то определить тела функций класса "КлассМассив".

```
DEFINE_LIST_TYPE ( Класс, КлассМассив )
или
DEFINE_LIST_BTYPE( Класс, КлассМассив )
```

Если "Класс" не содержит внутри себя захваченной памяти используйте DEFINE_LIST_BTYPE , а когда содержит используйте DEFINE_LIST_TYPE. Во втором случае класс "Класс" обязательно должен содержать функцию del(), чтобы не было утечек памяти.

Я это делаю, например, вот так:

```
// ------- декларация класса "массив long-ов" -----
DECLARE_LIST_TYPE( long, t_longList )

main()
{
   t_longList M ;     // описание переменной типа "массив long-ов"

   M.add(13);         // добавление в массив одного элемента
   i=M[0] ; M[0]=14 ; // использование массива
   ...
}
// ------- тело класса "массив long-ов" -----
DEFINE_LIST_BTYPE ( long, t_longList )
```

3.2 Класс t_Grammar

Класс t_Grammar (и все входящие в него) предназначен для храниения данных, прочитанных из файла описания грамматики lang.txt. Данные организованы по принципу "как слышится так и пишется", то есть каждый класс соответствует ровно одному понятию файла lang.txt. Будет полезно параллельно с чтением этой главы смотреть в файл readme.txt и в lang.txt.

Существует одна главная переменная Grammar типа t_Grammar, которая содержит в себе все данные, которые организованы в более мелкие структуры и массивы, которые в свою очередь могут содержать еще более мелкие структуры и массивы.

В качестве массивов я использую классы, сделанные на основе t_List. Очень удобно добавлять что-то в конец такого массива.

Сейчас для чтения и разбора файлов правилом хорошего тона является использование lex/yacc. Я и в этом случае опять не придерживаюсь правил хорошего тона и пишу разбор lang.txt на голом C++. Я знаю, и умею пользоваться lex/yacc-ом, но тут победило два соображения. Первое - "принцип не умножать сущности (в данном случае инструменты) без необходимости". То есть я не хотел, чтобы люди разбирающиеся в моей программе должны были знать что-то кроме C++. И второе - так исходники получались короче.

Итак описание t_Grammar и входящих в него классов в порядке от простого к сложному.

t_Value t_Param

Классы t_Value и t_Param изображают грамматический параметр языка. Вот как это выглядит в lang.txt:

```
@Параметр время   {  прошлое,настоящее,будущее  };
```

а вот как в C++:

```
// -------------- возможные значения параметров ------------------
struct t_Value
{  char Name[40] ;      // значение параметра например "Творительный"
} ;
// -------------- параметр слова - падеж и тд --------------------
struct t_Param
{
   char     Name[40] ;  // название параметра например "Падеж"
   t_ValueList Value ;  // возможные значения параметров
};
```

t_Param1

t_Param1 - это один параметр конструкции или ее составляющей. Все параметры конструкции - это массив t_Param1List. Вот так t_Param1List выглядит в lang.txt:

```
( род &Род, падеж Падеж, число Число )
```

а вот как в C++:

```
// -------------- параметр слова - падеж и тд --------------------
struct t_Param1
{  short param ;    // тип параметра в соответствии с Lang->Param
   char  Name[40] ; // название параметра например "Падеж"
   char  Dir ;      // 1-источник 0-приемник (есть & или нет)
   char  value ;    // значение параметра, если -1 значит не задан
};
DECLARE_LIST_TYPE( t_Param1,t_Param1List );
```

t_Form

t_Form - форма слова. Этой структуре нет соответствующего понятия в lang.txt. Вот ее описание на С++:

```
// ------------- форма слова ------------------------------------
struct t_Form
{  char value[10];  // порядок параметров - в соответствии с Part->Param
};
```

Как видите, всего навсего массив char из десяти элементов. Это значит, что любая конструкция языка может иметь не более 10 грамматических параметров. Немного, но достаточно (Если очень хочется, можете добавить). Если какая-то конструкция имеет параметры "род падеж и число", то нулевой байт - это значение рода, первый числа и т.д. Если параметров меньше десяти, лишние байты никак не используются. Значения параметров начинаются с нуля. То есть если Form.value[0]=0 - то это Именительный падеж. Form.value[0]=-1 то это значит, что параметр не задан или не вычислен.

t_Rename

t_Rename - псевдоним. Вот так это выглядит в lang.txt:

```
@Псевдоним  нс степени
```

а вот как в С++:

```
// ------------ переименование грамматических параметров ---------
struct t_Rename
{  char Reduce[6] ;  // сокращенное обозначение параметра (например "f")
   char Full[40] ;   // полное имя значения параметра (например "женский")
};
```

t_Format

Формат, описывающий грамматический словарь. Вот так это выглядит в lang.txt:

```
@Формат сущ формат2
{
   Число=Ед
   {  Падеж=И @Слово  Падеж=Р @Слово
      Падеж=Д @Слово  Падеж=В @Слово
      Падеж=Т @Слово  Падеж=П @Слово
   }
   Число=Мн
   {  Падеж=И @Слово  Падеж=Р @Слово
      Падеж=Д @Слово  Падеж=В @Слово
      Падеж=Т @Слово  Падеж=П @Слово
   }
}
```

а вот как в С++:

```
// ------------ формат, описывающий грамматический словарь -------
struct t_Format
{
```

```
   char      Name[40] ; // имя формата
   short     i_part ;   // индеск части речи в соответствии с
                        // t_Lang->Part[this->i_part]
   t_FormList Record ;  // список возможных форм слова
};
```

t_File
t_File файлы, в которых лежат грамматические словари. Вот так это выглядит в lang.txt:

```
@Файлы  глагол  формат1  { ru_verb  };
```

а вот как в C++:

```
// ----------- файлы, в которых лежат грамматические словари ---
struct t_File
{
   short  i_part,       // часть речи в соответствии с t_Lang->Part[this->i_part]
          i_format ;    // индекс формата
   t_strList  FileName ;// имена файлов, в которых лежит эта часть речи
};
```

t_PartDsk
t_PartDsk описание части речи в словаре переводов. Вот так это выглядит в lang.txt:

```
v  глагол ;
```

а вот как в C++:

```
// -------------- описание части речи в словаре переводов ------
struct t_PartDsk
{  char  Tag[2] ;    // тег (псевдоним) части речи в словаре переводов
   char  Name[40] ;  // имя части речи
   short i_part ;    // номер части речи
};
```

t_Format1
t_Format1 хранит всю информацию о формате файла перевода "от открывающей до закрывающей скобки". Вот так это выглядит в lang.txt:

```
@Формат_перевод формат1
{
   @Псевдоним  нс степени
   @Псевдоним  нв времени

   @Части_речи_источника
   {
      v  глагол ;
      m  мод_глагол ;
      b  связка ;
      s  сущ    ;
      a  прил   ;
      S0 гр_сущ_п0 ;
      S1 гр_сущ_п1 ;
```

```
      A0 гр_прил_п0 ;
      A1 гр_прил_п1 ;
      V1 фраза_пп ;
   }
   @Части_речи_приемника
   {
      v  глагол ;
      m  мод_глагол ;
      b  связка ;
      s  сущ    ;
      a  прил   ;
      S0 гр_сущ_п0 ;
      S1 гр_сущ_п1 ;
      A0 гр_прил_п0 ;
      A1 гр_прил_п1 ;
      V1 фраза_пп ;
   }
}
```

а вот как в C++:

```
// -------------- формат, описывающий словарь переводов -------
struct t_Format1
{  char Name[40] ;          // имя формата
   t_PartDskList SouPart,   // список переименования структур
                 DstPart ;
   t_RenameList  Rename ;   // список переименования параметров (псевдонимов)
};
```

e_Type
Энум e_Type - это тип конструкции.

```
// ----------- тип структуры ----------------------------------
enum e_Type
{  TNULL,     // нет в этом месте структуры - пустая запись
   TCONST,    // неизменяемый литерал                        @0[дурак]
   TCONST1,   // неизменяемый литерал из словаря             @1
   TWORD,     // слово
   TWORD0,    // пустой выбор                                @00
   TSTRUCT,   // структура                                   @Структура
   TSTRUCT1,  // фиксированная структура из словаря          @Структура1
   TSTRUCT2,  // произвольная структура из словаря           @Структура2
   TSELECT,   // выбор                                       @Выбор
   TENUM      // несимметричный выбор                        @Выбор1
} ;
```

t_Word
t_Word - слово, составляющая структуры, информация вот о такой записи:

```
гр_местоимения( &Род, Лицо, &Число, Падеж )
```

а вот как в C++:

```
// ----------------------- описание слова ------------------------
struct t_Word
{  e_Type type ;       // слово,структура или выбор
   char   Name[40] ;   // имя структуры например "глагол"
   char   literal[40] ;// строковая константа в квадратных скобках (это если
                // в lang.txt есть запись глагол[уразуметь] )
   t_Param1List Param; // параметры структуры
   short  order ;      // индекс упорядочивания например глагол<2>
   short  i_struct ;   // индекс структуры в массиве t_Grammar->Trans
   char   use ;
   char   meaning_use ; // признак того, что строковое значение передается наверх
                        // если есть восклицательный знак !глагол() )
};
```

t_Struct
t_Struct - информация о одной конструкции языка. Вот так это выглядит в lang.txt:

```
@Структура гр_сущ0( число Число ) =
           Наречие_степени
           заголовок
           опр_прилагательный
           опр_правый
           сущ( &Число  )
           хвост_сущ ;
```

а вот как в C++:

```
// ------------------- описание словосочетания ------------------
struct t_Struct
{  e_Type         type ;     // тип структуры
   char           Name[40] ; // имя структуры
   long           i_str ;    // строка, в которой содержится ее описание
                      // нужно, когда при чтении хочется ругнуться
   t_Param1List   Param ;    // параметры структуры
   t_WordList     Word ;     // составляющие
   t_RelationList Relation ; // соответствие параметров
}
```

Relation - соответствие параметров. Это массивчик, который строится уже после прочтения файла lang.txt. Каждый элемент этого массива, это указание того, как должны передаваться значение параметра (когда программа вычисляет параметры).
Одна Relation - это вот такая структура:

```
// ---------- направление передачи параметров внутри структуры -----
struct t_Relation
{  char s1,p1 ; // индекс структуры и параметра источника
   char s2,p2 ; // индекс структуры и параметра приемника
};
```

s1, s2 - номер составляющей. Если s1=0, - значит значение параметра передается из заголовка в составляющую, Если s2=0 - из составляющей в заголовок, если оба индекса не равны нулю, значит из одной составляющей в другую.

t_Table
t_Table - таблица трансляции параметров. Вот так это выглядит в lang.txt:

```
@Таблица( @Выбор, лицо Лицо = лицо Лицо )
{
   1  1е = 1е ;
   1  2е = 2е ;
   1  3е = 3е ;
   2  @0 = 3е ;
   3  @0 = 3е ;
}
```

а вот как в C++:

```
// ---------- элемент заголовка таблицы трансляции параметров -----
struct t_2Index
{  short i1,       // индекс слова, где этот параметр встречается со
                   // значком &
         i2,       // индекс параметра в структурк
         i_param ; // индекс параметра в языке
   char  Name[40]; // имя параметра, разумеется
};

struct t_Table
{  short        Size ;   // длинна строки таблицы (в данном случае 3)
   t_2IndexList In,Out ; // входные и выходные параметры
                         // в данном случае длинна их будет 2 и 1
   t_shortList  Value ;  // значения параметров
}
```

Для того, чтобы понять как и для чего используется эта таблица, прочтите соответствующую главу в readme.txt. Второй раз описать это я уже не могу.

t_Trans
Одна структура t_Trans - это информация о "трансляционной паре".

```
@Перевод
{
   @Структура гр_сущ0( число Число ) =
              Наречие_степени
              заголовок
              опр_прилагательный
              опр_правый
              сущ( &Число  )
              хвост_сущ ;
   @Структура гр_сущ0( род Род, падеж &Падеж, число Число ) =
              Наречие_степени
              заголовок( Род, Число, Падеж )
              опр_прилагательный( Род, Падеж, Число )
              сущ( &Род, Падеж, &Число )
              опр_правый
              хвост_сущ( Род, Падеж, Число ) ;
```

```
   table_chislo( Число )
}
```

а вот как в C++:

```
   struct t_Trans
   {  t_Struct    From,To ;       // описание структуры в соответствующем языке
      t_shortList Relation1,      // соответствие частей From->To
                  Relation2 ;     // соответствие частей To->From
      t_strList   Param1,Param2 ;// имена параметров той и другой структуры
      t_TableList Table ;         // таблицы трансляции параметров
   }
```

From и To - описание половинки трансляционной пары. Пусть вас не вводит в заблуждение имена From и To, это не те же From и To, которые были в t_Grammar.

Relation1 - соответствие частей структур. Вообще говоря порядок слов (и структур) на языке перевода и оригинала часто не совпадают. Например предлог, который в английской фразе стоит на последнем месте, при переводе надо ставить на первое место. Relation1 и Relation2. Это массивы отображающие соответствие порядка слов. Если Relation1[5] равно 1, это значит, что перевод пятого английского слова на первое место. К стати Relation2[1] должно быть равно 5.

t_Lang

t_Lang хранит всю информацию языка источника или языка приемника "от открывающей до закрывающей скобки":

```
   @Атомы_источника
   {  // ----------------- английский язык --------------------
      @Параметр число    {  Ед,Мн  };
      @Параметр род      {  М,Ж,С  };
      @Параметр лицо     {  1е,2е,3е  };
      @Параметр форма    {  Смысл,Inf,FormS,Form2,Form3,Ing  }; // форма глагола

      @Часть_речи глагол ( см_глагола &Смысл, форма Форма ) ;
      @Часть_речи мод_глагол ( форма Форма  ) ;
      @Часть_речи связка ( форма Форма  ) ;
      @Часть_речи сущ ( число Число  );
      @Часть_речи прил ( смысл_прил &Смысл );

      @Структура1 гр_сущ_п0 ( число Число ) = сущ( &Число ) ;
      @Структура1 гр_сущ_п1 ( число Число ) = @1 сущ( &Число ) ;

      @Структура1 гр_прил_п0 = @1 ;
      @Структура1 гр_прил_п1 = @1 сущ( &Число ) ;

      @Структура2 гр_глагола_пп( форма Ф ) = ;
      @Структура2 гр_have_пп( форма Ф ) = ;

      # include <format_a.h>

      @Файлы глагол           формат1  { en_verb };
      @Файлы сущ              формат2  { en_sub };
   }
```

вот как это представлено в C++:

```
// ------------- описание языка (источника или приемника) --------
struct t_Lang
{
   char          Name[40] ;
   t_ParamList   Param ;  // параметры
   t_StructList  Part ;   // части речи
   t_FormatList  Format ; // форматы грамматических словарей
   t_FileList    File ;   // файлы грамматических словарей
   t_Lexer      *Lexer ;
   char          To ;     // признак того, что это язых-приемник
};
```

t_Grammar
Класс t_Grammar - все данные, относящиеся к грамматике

```
class t_Grammar
{
   t_Lang         From ,To ; // язык оригинала и перевода
   t_Format1List  Format1 ;  // форматы переводов
   t_FileList     File1 ;    // имена файлов переводов
   t_TransList    Trans ;    // список всех трансляционных пар
   short          i_Number ; // часть речи, которая числительное
}
```

File1 - просто массив строк, каждая строка это имя файла, в котором содержится словарь перевода.

Можно заметить, что информация о части речи - это подмножество информации о "половинке"трансляционной пары. Точно так-же имеется имя и список параметров. По этому я посчитал, что удобно расположить информацию о частях речи там же, где лежит информация о трансляционных парах - в массиве Trans.

i_Number - индекс части речи "числительное". Эта переменная нужна для перевода тех случаев, когда числительное выражено не буквами, а цифрами.

3.3 Класс t_Slowo2

Класс t_Slowo2 служит для обслуживания грамматического словаря.

```
члены класса:
   t_Lang     *Lang ;   // язык, которому принадлежит словарь
   t_Format   *Format ; // формат словаря

   long        n_Word ; // число слов
   short       n_Form ; // число форм слова
   char       *Mass ;   // массив, в котором лежат слова (слова лежат
                        // друг за другом, разделенные пробелами)
   long       *Word ;   // ссылки на начала слов, длинна массива = n_Word*n_Form
   long       *reverce ;// алфавитный список форм, длинна массива = n_Word*n_Form

функции члены:
   t_Slowo2( void );
```

```
        конструктор,
деструктора нет, ибо не нужно

void    set_lf( t_Lang *_Lang, t_Format *_Format );
        Установить Lang и формат. Функция нужна для повторного чтения
данных (например при перепрочтении lang.txt).

void    read( char *File, t_Lang *Lang, t_Format *Format );
       Прочесть словарь из файла в соответствии с форматом.
File    - имя файла
Lang    - язык (источник или приемник), которому принадлежит словарь
_Format - формат, которому соответствует этот язык

void    freverce( void );
        Построить таблицы для поиска слов, заданных не в начальной форме

char   *normal( long index );
        Вернуть слово в начальной форме.
        index - индекс слова (строки)
Используется перед тем, как обратиться в словарь переводов.

char   *form( char *Str, t_Form *Form );
        Вернуть слово в заданной форме.
        Str   - слово в начальной форме
        Form1 - форма слова
Используется для формирования фразы на языке перевода.

short   quest( char *Str, long *Ant, t_Form *Form );
         найти слово и определить в какой оно форме
         Str   - искомое слово
         Ant   - индексы подходящих слов
         _Form - их формы
 Используется при разборе фразы на языке оригинала.

char    word_exist( char *str );
        Есть ли в словаре такое слово.

void    print_word( FILE *fw, long index );
        Напечатать все формы слова в заданный файл.
Используется для создания "секращенной версии" словаря.
private:
long    find( char *str );
        найти номер слова в словаре.

char   *form( long i, t_Form *Form );
        вернуть слово в заданной форме (другой интерфейс)
        index - индекс слова (строки)
        Form  - форма слова
```

3.4 Класс t_Slowo3

Этот класс обслуживает словарь перевода слов и выражений. Его структура настолько неотделима от алгоритма перевода, что его описание я даю вместе с описанием алгоритма перевода. Мне кажется так легче понять, что это такое, и как он работает.

3.5 Класс t_Lexer

Это вспомогательный класс, нужный только для того, чтобы прочесть текстовый файл, и разбить его на слова. Я думаю, бессмертные боги простят меня, если я не дам подробного его описания.

3.6 Класс t_Core

Самый главный класс, который занимается переводом. Его описание дается в главе «Синтаксический разбор»

4 Синтаксический разбор

4.1 Данные вообще

Вообще для перевода используются следующие данные.

```
t_Grammar    Grammar ;   // все данные, относящиеся к грамматике
t_Slowo3     Macro ;     // словарь английских сокращений (препроцессор)
t_Slowo3     Micro ;     // словарь русских сокращений (постпроцессор)
t_Slowo3     DicConst ;  // словарь английских constant
t_Slowo3     Perevod ;   // словарь переводов
t_Slowo2    *SlowoF,     // грамматические словари языка-оригинала
            *SlowoT ;    // грамматические словари языка-перевода
                         // массивы это, SlowoF[13] можно делать
short       *i_FormatT ; // соответствие часть речи -> SlowoT
short       *i_FormatF ; // соответствие часть речи -> SlowoF
                         // если i-я часть речи не имеет словаря,
                         // то i_FormatT[i]=-1 ;
t_Core       Core ;      // класс, занимающийся переводом
```

Это данные, которые описывают грамматику обоих языков, и их словари.

4.2 t_Core алгоритм и данные, общее представление

Главной трудностью при переводе является задача получить дерево структурного разбора переводимой фразы (См рисунок 1). Когда такое дерево построено, дальнейший перевод становится тривиальной задачей - нужно по этому дереву построить соответствующее дерево для русской фразы (возможно при этом какие-то конструкции будут переставлены), вычислить грамматические формы слов, и поставить русские слова в соответствующие формы. Все эти шаги достаточно легко (и даже рутинно) программируются.

А вот чтобы понять, какое слово играет какую роль, выделить синтаксические конструкции в довольно неряшливом речевом потоке человека, приходится потрудиться.

Перевод осуществляется следующим образом:

1. Фраза разбивается на слова (слово - это то, что между двумя пробелами).

Рис. 1: Дерево структурного разбора

```
                     +-----------+
                     |   фраза   |
                     +-----+-----+
                           |
              +------------+----------+----------------+
              |                       |                |
              V                       V                V
        +-----------+           +-----------+   +--------------+
        |  основа   |           |   хвост   |   |обстоятельство|
        +-----+-----+           +-----+-----+   +--------------+
              |                       |
      +-------+-------+               +---------------+
      |               |               |               |
      V               V               V               V
+-----------+   +-----------+   +-----------+   +-----------+
|подлежащее |   | сказуемое |   |дополнение1|   |дополнение2|
+-----------+   +-----------+   +-----------+   +-----------+
```

2. Далее препроцессор над этими словами производит (если необходимо) “незамысловатые преобразования”. Например «don’t» -> «do not». Просто удивительно насколько эти преобразования упрощают алгоритм перевода.
3. Производится определение какой частью речи и в какой форме является каждое слово.
4. Определяется в какой позиции какая конструкция может стоять, сколько вариантов имеет конструкция в каждой позиции, сколько слов занимает каждый вариант. Происходит заполнение главной таблицы (Взгляните на рис 2).
5. Производится попытка найти такие сочетания конструкций, которые все вместе образуют грамматически корректную фразу. Строится дерево вариантов (Взгляните на рис 3)
6. Для лучшего варианта (лучшим считается первый) строится "дерево структурного"разбора"фразы на языке источника. (Взгляните на рисунок 1).
7. Строится "дерево структурного разбора"фразы на языке приемника. На этом же этапе русским словам присваиваются строковые значения, то есть происходит перевод отдельных слов.
8. Вычисляются грамматические параметры. Производится вычисление параметров конструкций английской фразы, передача параметров в русскую фразу, вычисляются параметры слов русской фразы.
9. Русские слова ставятся в нужную форму, формируется русская фраза.
10. Постпроцессор выполняет “незамысловатые преобразования” над русскими словами. Например «бреет себя» -> «бреется».

Часть из этих пунктов просты до тривиальности. Их я описывать не буду. Я опишу только самое интересное, то что мне пришлось долго придумывать. Скажу по секрету, я придумал шесть принципиально разных алгоритмов перевода. Каждый следующий был чуть лучше предыдущего, понимал чуть больше. Нынешний, шестой - тоже не предел совершенства, но это первый, за который мне не стыдно.

Английский язык имеет дурацкий недостаток — каждое его слово может быть любой частью речи. По этому синтаксический разбор английской фразы выливается в попытку из тысяч (а то и миллионов) возможных конструкций собрать одну синтаксически верную фразу.

Чтобы справиться с этой задачей программе нужно знать:

1. В какой позиции какая конструкция может стоять.
2. Сколько вариантов имеет в этом месте эта конструкция.
3. Из каких частей состоит этот вариант конструкции.
4. Какой вариант составляющих использует данный вариант конструкции.
5. Грамматические параметры конструкции
6. Смысл (главное слово) данной конструкции. (Ну типа надо знать, что в сказуемом "must begin to work", значащий глагол - "work")

Существует т.н. "главная таблица вариантов конструкций", которая отвечает на вопрос "какая конструкция в какой позиции может стоять". Смотри Рис 2 Эта таблица в свернутом виде содержит в себе все варианты перевода фразы. (Очень свернутом, вообще говоря их могут быть миллионы, я при отладке наблюдал такие случаи.)

Конечно, «в действительности все не так, как на самом деле», реализация этой таблицы не столь прямолинейна, как это нарисовано (мне приходилось, экономить память, и пускаться на ухищрения), Но эта модель должна быть в голове, пока Вы читаете этот текст.

Что можно увидеть на этом рисунке. Столбцы таблицы - это слова. Строки таблицы - это конструкции. Цифры в клетках - это количество вариантов конструкции, которые начинаются с этого слова. Если цифра в клетке на стоит, значит конструкция с этого слова начинаться не может.

Ниже таблицы нарисована одна клетка таблицы «в увеличенном масштабе». Нарисовано, что фраза_пп в каком-то месте имеет три возможных варианта. Варианты, разумеется, должны чем-то отличаться друг от друга. Они отличаются составляющими, в данном случае сказуемыми и дополнениями. То есть фраза_пп вариант 1 и 2 имеет сказуемое вариант 1 а фраза_пп вариант 3 имеет сказуемое вариант 2.

После того, как заполнена Главная таблица, можно попытаться собрать корректную конструкцию. Допустим в результате заполнения главной таблицы сложилась следующая ситуация:

1. Допустим у нас имеется структура (фраза) в которую входят три составляющие (подлежащее, сказуемое и дополнение)
2. Подлежащее может начинаться с первого слова и иметь в своем составе одно два или три слова. (Допустим что, как всегда, существительные с глаголом различить невозможно)
3. Сказуемое, если оно начинается со второго слова, может иметь в своем составе 2 и 1 слово, если оно начинается с третьего слова, то только одно слово, и не может начинаться с четвертого.
4. Дополнением может быть второе и четвертое слово.

То есть если мы приняли решение, что подлежащее занимает 1 слово, то продолжать фразу можно двумя способами, Если подлежащее занимает 2 слова, то только одним, а если 3 слова, то продолжить фразу невозможно. Ну и так далее. Таким образим получается "дерево вариантов"построения конструкции. (Смотрите Рис 3).

Конструкцию можно корректно построить, если в дереве вариантов есть ветка, содержащая все нужные составляющие структуры (в данном случае три). Если таких веток несколько, значит фразу можно перевести несколькими способами.

4.3 t_Core формальное описание данных

t_Core главный класс перевода (в основном главная таблица).

```
// ---------- главный класс перевода -------------------------
class t_Core
{
   // ----- исходная фраза во всяко-разных представлениях ------------
```

Рис. 2: Главная таблица вариантов конструкций

```
                   Слова

             You      will     back     them     up.
Структуры   +--------+--------+--------+--------+--------+
сущ         |        |        |1       |        |        |
            +--------+--------+--------+--------+--------+
местоимение |1       |        |        |1       |        |
            +--------+--------+--------+--------+--------+
глагол      |        |1       |1       |        |1       |\
            +--------+--------+--------+--------+--------+ +
сказуемое   |        |1       |        |        |1       |\|
            +--------+--------+--------+--------+--------+ +
ядро        |1       |        |        |        |        |\|
            +--------+--------+--------+--------+--------+-+
хвост       |        |        |2       |1       |        |
            +--------+--------+--------+--------+--------+
фраза_пп    |1       |        |        |        |        |
            +--------+--------+--------+--------+--------+
фраза       |4       |1       |        |        |1       |\
            +--------+--------+--------+--------+--------+ +
             \вариант1\вариант1\|                \вариант1\|
              +--------+--------+                 +--------+
               \вариант2\|\
                +--------+ +
                 \вариант3\|\
                  +--------+ +
                   \вариант4\|
                    +--------+


             Одна клетка таблицы

фраза_пп                        составные части структуры фраза_пп
+--------------------+     +--------------+-------------+--------------+
| фраза_пп вариант 1 |---->| подлежащее 1 | сказуемое 1 | дополнение 1 |
+--------------------+     +--------------+-------------+--------------+
| фраза_пп вариант 2 |---->| подлежащее 1 | сказуемое 1 | дополнение 2 |
+--------------------+     +--------------+-------------+--------------+
| фраза_пп вариант 3 |---->| подлежащее 1 | сказуемое 2 | дополнение 2 |
+--------------------+     +--------------+-------------+--------------+
```

Рис. 3: Дерево вариантов структуры

```
+----------+
|  фраза   |
+----------+
  |
  |  +----------+    +------------+
  +->|подлежащее|-+->|сказуемое(1)|-+---------------------------------------------->
     +----------+ |  +------------+ |
     | мест You | |  |глагол will | |  +--------------+
     +----------+ |  +------------+ +->|обстоятельство|---------------------------->
                  |                 |  +--------------+
                  |                 |  | наречие back |
                  |                 |  +--------------+
                  |                 |
                  |                 |  +------------+
                  |                 +->|  хвост (1) |------------------------------>
                  |                 |  +------------+
                  |                 |  |  сущ back  |
                  |                 |  +------------+
                  |                 |
                  |                 |  +--------------------+
                  |                 +->|      хвост (2)     |-+-------------------->
                  |                    +---------+----------+ |
                  |                    |сущ back | мест them| |
                  |                    +---------+----------+ |  +--------------+
                  |                                           +->|обстоятельство|->
                  |                                              +--------------+
                  |   +------------------+                       |  наречие up  |
                  +-->|   сказуемое (2)  |-+-----------------+   +--------------+
                      +------+-----------+ |                 |
                      | will |глагол back| |  +-----------+  +-------------------->
                      +------+-----------+ +->|  хвост(3) |
                                              +-----------+
                                              | мест them |-+-------------------->
                                              +-----------+ |
                                                            |  +--------------+
                                                            +->|обстоятельство|-->
                                                               +--------------+
                                                               |  наречие up  |
                                                               +--------------+
```

```
   char         Sou0[1000] ;   // исходная фраза разбитая на слова
   char         Sou1[1000] ;   // исходная фраза не разбитая на слова
   char         Sign ;         // знак в конце предложения '.' '!' или '?'
   t_ItemList   From ;         // структуры From

   // ----- главная таблица ------------------
   t_Variants  *Variants ;        // клетки главной таблицы
   short        n_struct,n_word ; // размерность главной таблицы
   t_rWordList  rWord ;           // составляющие структур

   // ----- смыслы Конструкций ---------------
   char        *Meaning ;      // Массив, хранящий строки
   long         j_Meaning,     // указатель на первый свободный байт
                l_Meaning ;    // общая длинна массива Meaning

   // ----- варианты переведенной фразы -------
   t_Antwort   *Antwort;       // массив вариантов
   long         n_Antwort ;    // число вариантов перевода
   char         f_Full ;       // признак, что существуют полные варианты
}
```

Главная таблица.

Главная таблица - это массив t_Variants *Variants. Размерность этого массива n_struct*n_word. Этот массив захватывается для каждой фразы, и после перевода освобождается. Массив внутренний, извне класса к нему доступа нет, да и изнутри не приветствуется (легко ошибиться и выйти за размеры массива).

Для того чтобы получить список вариантов i-й конструкции, начинающихся с j-го слова есть функция:

```
   t_Variants *variants( short i_word, long i_struct );
```

Для того, чтобы получить k-й вариант этой конструкции есть функция:

```
   t_rStruct  *variant(  short i_word, long i_struct, long i_variant );
```

t_Variants - одна клетка главной таблицы.

```
// --------- все возможные варианты реализации структуры ----
struct t_Variants
{  char  absend ;          // признак того, что эта конструкция в
                           // этой позиции стоять не может
   short i_struct,i_word ; // индекс строки (структуры) и столбца (слова)
                           // вообще-то это дублирование информации, потому что
                           // по индексу в массиве Core.Variants можно все вычислить
   // и i_struct и i_word но оочень удобно хранить эти
                           // индексы в самой структуре
   t_rStructList Variant ; // массив вариантов реализации структуры в этом месте фразы
};
```

t_rStruct вся имеющаяся информация о варианте раелизации структуры.

```
// ----------- вариант реализации структуры ------------------
struct t_rStruct
{
   e_Type  type ;
   short   i_word,       // с какого слова начинается
           i_last_word,  // с какого слова начинается следующая
           i_struct ;    // номер структуры в t_Grammar->Trans[]
   long    r_word ;      // ссылка на t_Core->rWord
   long    i_slowo ;     // индекс в словаре переводов
   short   i_slowo1 ;    // способ перевода
   t_Form  Form ;        // грамматические параметры этого варианта
   long    i_meaning ;   // смысл этой конструкции на языке оригинала
                         // (указывает на t_Core->Meaning)
   t_Quality Q ;         // "качество" варианта
};

// ----------- вариант реализации слова ----------------------
struct t_rWord
{
   e_Type  type ;
   short   i_word,      // номер слова, с которого начинается
           i_last_word, // номер слова, с которого начинается следующий
           i_struct;    // номер структуры в t_Grammar->Trans[]
   long    i_variant;   // номер варианта перевода
   short   index ;      // индекс в массиве t_Struct->Word[]
   long    i_slowo ;    // индекс в словаре переводов
   short   i_slowo1 ;   // способ перевода
   short   n_slowo ;    // число вариантов перевода
};
```

Составляющие варианта реализации конструкции лежат в массиве rWord начиная с rStruct[i].r_word. Число составляющих в rStruct не содержится. Его надо брать из Grammar.Trans[rStruct[i].i_struct].Word.j .

Смысл (главное значащее слово) конструкции можно узнать с помощью функции

```
char  *get_meaning( t_rStruct *V );
```

и установить с помощью

```
void  set_meaning( t_rStruct *V, char *Str );
```

Даже я не пытаюсь делать это напрямую, потому что легко запутаться.

t_Antwort - вариант синтаксического разбора фразы со всеми возможными данными о нем - деревом синтаксического разбора источника и приемника, указанием того, сколькими способами можно перевести каждое слово, ссылками из какой клетки главной таблицы получена каждая конструкция и т.п.

```
class t_Antwort
{
   char         Str[200] ;     // собственно сторока ответа
   t_TreeList   Tree1, Tree2 ; // структурное дерево фразы-источника и фразы приемника
   t_aWordList  aWord ;        // массив, который никак не участвует в процессе
```

```
                                   // перевода, но зато очень удобен, когда из 20 значений
                                   // слова нужно выбрать одно
}
```

t_Tree и t_TreeList Дерево структурного разбора фразы. Невероятно, но факт. Эта структура оказалась удобной для описания двух очень разнородных сущностей - дерева структурного разбора фразы (Рис1) и дерева вариантов фразы (Рис2). Правда во втором применении используются не все поля.

```
struct t_Tree
{  e_Type type ;        // тип узла
   long   up ;          // ссылка вверх на родительскую вершину
   long   down,n_down ; // ссылка вниз на первого потомка, и число потомков
   short  i_word,       // с какого слова начинается эта структура
          i_last_word,  // с какого слова начинается следующая
          i_struct ;    // что за конструкция (можно обратиться в
                        // Grammar.Trans[this->i_struct] и узнать все про эту
                        // конструкцию
   long   i_variant ;   // номер варианта структуры (если главаная таблица
                        // считает, что в этом месте эта структура может быть
                        // реализована несколькими разными способами)
   long   link ;        // ссылка вершины источника на соответствующую вершину
                        // приемника или наоборот
   t_Form Form ;        // значения грамматических параметров в соответствии
                        // с Struct->Param
   short  select ;      // выбранный вариант в TSELECT (актуально только для TSELECT)
   short  index ;       // номер слова в структуре t_Struct->Word
                        // текщая конструкция всегда является частью какой-то
                        // более крупной конструкции. Это поле - номер этой
                        // конструкции в списке составляющих частей родителя
   long   i_slowo ;     // индекс Record в словаре переводов
                        // актуально для STRUCT1 и STRUCT2
   short  i_slowo1 ;    // способ перевода (какое русское значение выбрано)
   short  n_slowo  ;    // число вариантов перевода (сколько значений имеет
                        // это слово не русском языке) актуально для слов
                        // и "конструкций из словаря" и STRUCT1 и STRUCT2
   t_Quality  Q ;
   char   Str[50] ;     // строковое значение в нужной форме
   char   Str1[50] ;    // строковое значение в начальной форме
} ;
```

t_aWord - слово в переведенной фразе и все его характеристики грамматические и смысловые.

```
struct t_aWord
{  short  i_struct ;// это как всегда
   long   i_slowo ; // это тоже
   short  n_slowo ; // число вариантов перевода
   t_Form Form ;    // грамматическая форма
   char   Str[50] ; // строка (нужна для служебных слов, когда слово в переведенной
                    // фразе не является результатом перевода соответсвующего
                    // английского слова)
};
```

4.4 t_Core формальное описание алгоритмов

Заполнение главной таблицы производит функция t_Core::universe(). Функция имеет два параметра i_struct - номер структуры (номер строки) и i_word - - номер слова (номер колонки). Эта функция рекурсивная, то есть в процессе работы она может вызвать сама себя.

Конструкции бывают двух основных типов - «Выбор» и «Структура» (с вариациями, разумеется, но вариации сейчас не важны).

С выбором все просто. Список вариантов выбора - это простая сумма списков вариантов входящих в него конструкций. Например у меня есть такая конструкция типа «выбор» - «ядро», в который входят «ядро повествовательное», «ядро модальное», «ядро вопросительное» и т.д. Допустим повествовательное имеет 2 варианта реализации, модальное 1 и вопросительное 0. Ясно, что ядро будет иметь 3 варианта реализации.

А вот если конструкция имеет тип "Структура это таакая песня!...

Алгоритм построения дерева вариантов конструкции. Прежде чем читать дальше еще раз посмотрите на Рис 3.

1. Внесение нулевой вершины.

2. Произвести обход всех вершин дерева в прямом направлении, с достраиванием дерева в процессе обхода.Для каждой вершины сделать следующее:

   (a) Определить сколько вариантов есть в клетке, соответствующей этому номеру слова этому номеру конструкции.

   (b) Если ни одного, перейти к следующей вершине.

   (c) Если клетка еще не заполнена (неизвестно количество вариантов этой конструкции), вызвать этот алгоритм построения вариантов (universe) для этой клетки .

   (d) Если это клетке соответсвует конструкция типа «Структура1» или «Структура2» - заполнить эту клетку с помощью функции t_Slowo3::s_universe. (Эта функция аналогична universe с той лишь разницей, что она работает со структурами из словаря.)

   (e) Внести все варианты из этой клетки в качестве сыновей текущей вершины. Присвоить сыновьям их характеристики.

3. Произвести собирание полных вариантов, для этого произвести обход всех вершин дерева в прямом направлении.

   (a) Если мы встретили вершину, ранг которой равен числу составляющих структуры (удалось найти все составляющие структуры), то эта вершина соответсвует найденному варианту. Если ранг вершины меньше, то она должна быть пропущена.

   (b) Вписываем в главную таблицу найденный вариант структуры. Ясно, что составляющие варианта лежат вдоль пути, приведшего нас в эту вершину. Было бы очень удобно вписать все составляющие двигаясь из этой вершины к началу дерева, по ссылкам на родителя. Но ведь в этом случае части запишутся в обратном порядке! По этому делаем это в два прохода.

   (c) Вписываем в массив составляющих N_Part пустых составляющих.

   (d) Двигаясь по ссылкам на родителя, записываем характеристики составляющих в массив rWord. (Ну тут с индексами приходится мудрить.)

К стати, а вы поняли, что эта программа может быть зациклена, если Вы невнимательно описывали грамматику? Если у вас есть две структуры, первая в качестве составляющей включает вторую, а вторая первую, и обе являются первой составляющей, то программа проваливается в бесконечный цикл.

Рис. 4: Взаимосвязь между структурами, образующими класс t_Slowo3

```
           массив Mass
               ---+------------------------------------------+----
           ...    |S1:@0[binary] s[code]=S1:a[двоичный] s[код]|    ....
               ---+------------------------------------------+----
                         A         A          A          A
                         |         |          |          |
                         |         |          |          +-----------+
                         |         |          +--------------------+ |
                         |         +----------------------------+  | |
                         +------------------------------------+ |  | |
                                                              | |  | |
                                                              | |  | |
 массив Record        массив Struct           массив Word     | |  | |
+--------------+     +--------------+       +--------------+  | |  | |
|     sy_struct|---->|       sy_word|------>|           str|--+ |  | |
+--------------+     +--------------+       +--------------+    |  | |
|              |-+   |              |----+  |              |----+  | |
    .......      |   +--------------+    |  +--------------+       | |
                 +-->|              |-+  +->|              |-------+ |
                        ........      |     +--------------+         |
                                      |     |              |---------+
                                      |     +--------------+
                                      +---->|              |
                                               ........
```

4.5 Работа с выражениями Класс t_Slowo3

Этот класс предназначен для словаря переводов. Словарь содержит не только слова, но и словосочетания, причем иногда весьма причудливые. По этому данные этого класса довольно сложные.

Посмотрите на файл base_enru любым текстовым редактором. Убедитесь в том, что вы понимаете то, что там написано.

Основные данные класса t_Slowo3 следующие:

1. Mass - это просто массив, содержащий файл base_enru в памяти.
2. Record - кусок данных, соответствующих одной строке base_enru, он же одно английское выражение, и несколько его переводов.
3. Struct - одно выражение русское или английское. Один Record содержит один английский Struct, и один или более русских Struct-ов.
4. Word - слово входящее в выражение. Один Struct содержит одно или несколько Word-ов.

См. рис 4 Эх, ностальгия! Раньше такие плакаты, показывающие взаимосвязи между структурами висели вдоль стены какого-нибудь НИИ. Сейчас такого уже не увидишь...

Мой стиль программирования несколько старомоден. Сейчас в моде контейнеры, содержащие в себе другие контейнеры, и так далее. Я пишу в стиле Кернигана- Ричи. У меня есть четыре массива, для каждого типа структур. Связи между массивами задаются специальными полями-индексами (каждый индекс - это целое число). Например Record[i].sy_struct - это индекс первой структуры, относящейся к i-му Record. То есть к i-му Record относятся все структуры

Рис. 5: Структура (и смысл) дерева t_Slowo3

```
 голова       1 ярус              2 ярус           3 ярус    4 ярус   Record

+-------+   +----------+                                          +--+
|голова |-->|глаголы   |----------------------------------------->|  | make
+-------+   +----------+                                          |  | take
            |сущ       |----------------------------------------->|  | program
            +----------+      +--------+                          |  | computer
            |гр_сущ_пп |->[]-->| сущ    |------------------------->|  | many
            +----------+      +--------+      +------+            |  |
            |гр_прил_пп|-+     | прил   |->[]->| сущ  |----------->|  | small money
            +----------+ |     +--------+      +------+   +-----+ |  | big   money
                         |                     | прил |-->| сущ |->|  | small small money
                         |                     +------+   +-----+ |  | big big money
                         |     +--------+                         |  |
                         +-[]->| прил   |------------------------->|  | beautiful
                               +--------+      +------+           |  |
                               | наречие|----->| прил |----------->|  | very beautiful
                               +--------+      +------+           |  |
                                                                  +--+
```

с индексами от Record[i].sy_struct до Record[i+1].sy_struct. Надо ли обьяснять, зачем массив Record надо захватывать длинной n_Record+1? Когда данные статические, такой способ размещения данных более компактен, обеспечивает меньшее время доступа, ну и программировать легче.

Кроме того, есть такая структура - дерево (См рис 5), которая позволяет найти в словаре выражение по критерию. Критерий простой - соответствует ли ему переводимая фраза.

Дерево построено следующим образом -

1. Нулевая вершина - корень дерева.
2. Первый ярус вершин (сыновья нулевой вершины) - типы структуры. То есть все выражения имеющие один тип структуры (например гр_сущ_пп) - будут потомками одной вершины.
3. Второй ярус вершин - это тип первого слова выражения. То есть все выражения имеющие один тип структуры гр_сущ_пп, и первое слово которых будет прилагательным будут сыновьями одной вершины второго яруса. Внутри вершины выражения упорядочены по алфавиту.
4. Второй ярус вершин - это тип второго слова выражения. И так далее.

Пустые квадратики - это "пустые вершины". Я их вставляю в дерево если у вершины есть сыновья с разными типами конструкций (например существительное и прилагательное).

Обратите внимание, что по построению у всех Record, являющихся потомками одной вершины i-го уровня, i первых слов будут одинаковые. Это свойство используется при поиске подходящих выражений.

Массив Record на 5, это тот же самый Record, что и на рисунке 4. Вы можете мысленно состыковать эти два рисунка. Я бы сам это сделал, но ширина страницы не позволяет.

4.6 Как происходит поиск выражения в t_Slowo3

t_Core и t_Slowo3 - близнецы-братья! Кто более для программы Меркурия ценен? Мы говорим t_Core - подразумеваем t_Slowo3, мы говорим t_Slowo3 подразумеваем t_Core.

Функции s_universe дается задание - найти (или сказать, что не найдено) такую-то структуру в такой-то позиции. s_universe просматривает имеющиеся в словаре структуры, и складывает подходящие в главную таблицу.

Программа должна пройти по дереву словаря, и найти все Record-ы типа i_struct, которые могут начинаться со слова i_word. Для того, чтобы это сделать, строится "дерево поиска", которое является подмножеством дерева словаря. Включается или исключается соответствующая ветка дерева словаря в дерево поиска, определяется тем, есть ли в главной таблице нужная структура (Например можно ли второе слово фразы интерпретировать как глагол, и существует ли Record - заданного типа, у которого второе слово - глагол "will"). Если двигаясь таким образом удается дойти до финальных вершин дерева, соответствующие Record-ы помещается в главную таблицу.

Поиск осуществляется следующим образом:

1. Производится поиск среди вершин первого яруса. Если вершины с нужным типом структуры нет, значит такую структуру найти невозможно, и делается выход.

2. Если вершина нашлась, начинается строиться "дерево поиска"(массив Node).

   Дерево строится примерно также как оно строилось в t_Core::universe() (Правда есть некоторые отличия.)

3. Дерево Node обходится в прямом направлени. В каждый момент у нас есть "текущая вершина"дерева вариантов Node[i], и "текущая вершина"дерева словаря Tree[i_tree], которая соответствует "текущей вершине"дерева вариантов.

   (a) Если текущая вершина Tree[i_tree] - "пустая"То в дерево вариантов вносится вершины, соответствующие сыновьям Tree. после чего происходит переход к следующей вершине Node.

   (b) Если вершина Tree[i_tree] имеет тип константа, find_bin() смотрит есть ли среди ее потомков Record, которая содержит в себе в нужном месте нужную константу.
   Например: Разбирается фраза "I will take part in ..."
   Допустим, у нас есть две Record такого вида:

   ```
   VV:v1[take] @0[down]=VV:v1[разрушать]
   VV:v1[take] @0[part]=VV:v1[принимать] @0[участие]
   ```

   И очередная вершина соответствует второму уровню Tree, и четвертому словы фразы.
   В этом случае в дерево вариантов будет внесена вершина, второму Record.

   (c) Если вершина Tree[i_tree] является конструкцией, определяется сколько конструкций такого вида распознано в этой позиции. (Сколько вариантов конструкции такого вида возможно в этой позиции)
   Если t_Core еще не заполнило эту клетку главной таблицы, то вызывается t_Core::universe(), которая заполняет эту клетку главной таблицы.
   Далее для каждого варианта определяется, соответствует ли он какому-нибудь Record в словаре. Если да, в дерево Node записывается очередная вершина.
   Например Разбирается фраза "I will take part in ..."
   Допустим у нас есть две Record такого вида:

   ```
   VV:v1[take] @0[part]=VV:v1[принимать] @0[участие]
   VV:v1[talk] @0[out] =VV:v1[исчерпывать] @0[тему]
   ```

И Core дает два варианта сказуемого - "will"и "take". (В первом варианте will распозналось как глагол, во втором - как служебное слово) В этом случае "will"не даст никакого продолжения (соответсвующих Record нет), а "take"внесет в дерево вариантов вершину, соответсвующую первому Record.

4. Далее происходит собирание вариантов. Если в дереве вариантов Node, есть конечные вершины (Которые соответствуют конечным вершинам Tree) то такие Record должны быть занесены в главную таблицу t_Core. Если таких вершин нет, это значит, структура искомого типа не может стоять в данном месте фразы.

   (a) Происходит заполнение заголовка структуры

   (b) Внесение (но не заполнение) составляющих структуры

   (c) Заполнение "задом наперед"составляющих структуры. Задом наперед, потому что так легче пройти по нужным ветвям дерева вариантов - каждая вершина дерева Node имеет ссылку на родителя.

4.7 Как t_Slowo2 хранит свои данные

Грамматические словари программы «Меркурий» существуют в двух видах - полном и сокращенном. В полном виде словарь выглядит как массив записей следующего вида:

```
победа победы победе  победу победой  победе
победы побед  победам победы победами победах
;
```

Вообще говоря, это очень расточительный способ записи. Эти же данные можно хранить гораздо компактнее в виде слова и номера шаблона его словоизменения. Например вот так:

```
@Шаблоны
{
  ...
  4 3134 -а -ы -е -у -ой -е -ы - -ам -ы -ами -ах ;
  ...
}
победа 4 ;
```

или вот так:

```
@Шаблоны
{
   0   6344 -ть -ю -ешь -ет -ем -ете -ют -л -ла -ло -ли ;
   3   1349 -ить -ю -ишь -ит -им -ите -ят -ил -ила -ило -или ;
   410 9805 - -а -о -ы ;
   415 3943 -ий -ая -ее -ие -его -ей -его -их -ему -ей -ему -им -его -ую
            -его -их -им -ей -им -ими -ем -ей -ем -их ;
   426 3921 -ся -тесь ;
   443 0 - ;
}
побеждать   0
побежден    410
побеждай    426
побеждающий 415
```

```
победавший 415
побеждая    443
победить    3
побежден    410
победи      426
победивший  415
победив     443
;
```

В данном случае (для русских глаголов) используется "множественный шаблон". То есть для действительного и страдательного залога, повелительной формы, причастия и деепричастия одного и того же глагола используются свои разные шаблоны.

Класс t_Slowo2 обеспечивает хранение грамматического словаря и в полной, и в сокращенной форме. Интерфейс доступа к данным в обоих случаях одинаковый.

В случае хранения в полной форме используются следующие массивы данных:

```
long        n_Word ; // число слов
short       n_Form ; // число форм
char       *Mass ;   // Массив, в котором лежат слова
long       *Word ;   // ссылки на начала слов [n_Word*n_Form]
long       *reverce ;// алфавитный список форм
```

```
   Mass
   --+--------+--------+--------------------+-----------+---
     | победа | победы | победе ... победах | победитель|  ...
   --+--------+--------+--------------------+-----------+---
       A        A        A          A         A
       |        |        |          |         |
       |        |        |          +-----+   |
       |        |        +--------------+ |   |
       |        +---------------------+ | |   |
       +----------------------------+ | | |   |
                                    | | | |   |
 reverse                 Word       | | | |   |
+---------+            +---------+  | | | |   |
|         |----------->|         |--+ | | |   |
+---------+            +---------+    | | |   |
|         |------+  +->|         |----+ | |   |
+---------+  +---|--+  +---------+      | |   |
|         |--+   +---->|         |------+ |   |
+---------+            +---------+        |   |
|         |            |         |        |   |
   ...                    .....           |   |
                       |         |--------+   |
                       +---------+            |
                       |         |------------+
                       +---------+
                       |         |
           ....
```

То есть, если мы знаем, что слово "Победа"у нас имеет номер 1000, и мы хотим получить форму именительный падеж, множественное число, то строка представляющая эту форму лежит в Mass+Word[1000*n_Form+6]. Здесь 1000 - номер слова, 6 - номер формы.

Массив reverce содержит ссылки на Word[] в алфавитном порядке. Он используется, для бинарного поиска, если у нас есть слово (например "победы") и мы хотим узнать номер этого слова и номер формы.

В случае хранения в сокращенной форме используются следующие массивы данных:

```
t_Shabl    *Shabl ;   // массив шаблонов
t_compWord *Word1 ;   // словоформа + номер_шаболна
long       *sy_Word1 ;// ссылки на Word1
long        n_Shabl ; // число шаблонов
long        n_Word1 ; // число слов
long       *reverce ; // алфавитный список форм
```

Здесь все не так просто как полной форме.

```
         Shabl
      +-----+------+-----------------------------------------------------+
+----->| 0   | 6344 | -ть -ю -ешь -ет -ем -ете -ют -л -ла -ло -ли ;       |
|      +-----+------+-----------------------------------------------------+
|      | 3   | 1349 | -ить -ю -ишь -ит -им -ите -ят -ил -ила -ило -или ;  |
|      +-----+------+-----------------------------------------------------+
| +--->| 410 | 9805 | - -а -о -ы ;                                        |
| |    +-----+------+-----------------------------------------------------+
| |    | 415 | 3943 | -ий -ая -ее -ие -его -ей -его -их -ему -ей -ему ... |
| |    +-----+------+-----------------------------------------------------+
| | +->| 426 | 3921 | -ся -тесь ;                                         |
| | |  +-----+------+-----------------------------------------------------+
| | |  | 443 | 0    | - ;                                                 |
| | |  +-----+------+-----------------------------------------------------+
| | |
| | +-----------------------------------------+
| +-----------------------------------------+ |
+-----------------------------------------+ | |
                                          | | |
     sy_Word1               Word1         | | |
    +-----------+      +-----------------+  | | |
    |           |----------->| побеждать   0   |--+ | |
    +-----------+      +-----------------+    | |
    |  |-----+      | побежден    410 |----+ |
    +-----------+     |      +-----------------+      |
                      |      | побеждай    426 |------+
                      |      +-----------------+
                      |      | побеждающий 415 |----
                      |      +-----------------+
                      |      | побеждавший 415 |
                      |      +-----------------+
                      |      | побеждая    443 |
                      |      +-----------------+
                      |      | победить    3   |
                      |      +-----------------+
```

```
|       | побежден    410 |
|       +-----------------+
|       | победи      426 |
|       +-----------------+
|       | победивший  415 |
|       +-----------------+
|       | победив     443 |
|       +-----------------+
+----->| побивать        |
           ...
```

Shabl - Шаблон, массив из нескольких окончаний слова. Word1 - словоформа + номер_шаболна, например "побеждающий 415"sy_Word1 - ссылки на первую словоформу соответсвующего слова. В простейшем случае, когда нет множественных шаблонов, в этом массиве содержатся целые числа по порядку - 0,1,2,3... То есть одному слову соответствует ровно один шаблон. В случае множественных шаблонов в этом массиве содержится что-то типа 0,12,24,36... В данном случае одному слову соответствует 12 словоформ с их шаблонами. reverce - алфавитный список форм.

Теперь, если у нас есть слово "победить"и нам нужно получить, например, причастие совершенного вида, такой-то род, такой-то падеж, то нужно 1) найти это слово, 2) найти эту словоформу 3) по шаблону соответствующему словоформе образовать, то что надо (причастие).

reverce - содержит алфавитный список форм. Но хитрость заключается в том, что у нас нет готовых форм слова. Мы по-прежнему реализуем бинарный поиск, но перед каждым сравнением создаем нужную форму нужного слова.

Теперь несколько слов о том, как формируется массив reverse в случае сжатого словаря. Можно было бы просто создать список всех форм слов, и потом просто применить qsort. Но мы боролись за экономию памяти, и не хотели одновременно создавать формы всех слов.

Для этого была написана функция freverse2. Можно заметить, что форма слова строится так: к постоянной основе слова прибавляется соответсвующее окончание. Таким образом упорядочив по алфавиту окончания внутри шаблона, можно упорядочить формы принадлежащие одному слову. То есть мы будем знать, что сначала идет Имен. падеж Ед число, потом Дат. Мн., потом Тв. Мн. и т.п. Этот этап отмечен в программе комментарием "сортировка внутри шаблонов".

Далее производится сортировка слов по их "самой первой с точки зрения алфавита"форме. Этот этап отмечен комментарием "сортировка сжатых слов".

Далее производится сортировка слиянием. У нас есть несколько массивов. Word1 Массив отсортированных сжатых слов. Tmp0 Массив разжатых слов. Этот массив содержит все формы очередного слова. Tmp1 Временный массив разжатых слов, которые уже вышли из массива Word1, но еще не попали в reverce. Tmp2 Второй временный массив разжатых слов reverce формируемый массив номеров форм.

Первоначально массивы Tmp1 и Tmp2 пусты. Tmp0 заполняется формами очередного слова.

Мы сливаем два массива Tmp0 и Tmp1 (оба отсортированы по алфавиту). На каждой итерации цикла мы сравниваем очередной элемент из того и другого массива, и выбираем лексографически меньший. Если этот элемент меньше, чем следующее слово из массива Word1, то элемент заносится в выходной массив reverce, если больше, то в Tmp2.

Когда Tmp0 и Tmp1 подойдут к концу, Tmp0 заполняется формами следующего слова, содержимое Tmp2 переписывается в Tmp1, Tmp2 делается пустым (и таким образом поддерживается существование отсортированного временного массива).

5 Построение ответа

5.1 Построение Дерева фразы источника

А вот теперь свернутое представление варианта перевода, содержащееся в главной таблице, нужно развернуть, и получить дерево структурного разбора, показывающее строение фразы (См рис 1).

Алгоритм разворачивания такой:

1. Вносится первая вершина. Первая вершина - это i-й вариант структуры типа "все_все".

2. Производится обход дерева-источника в прямом направлении, с достраиванием дерева в процессе обхода. На каждом шаге в дерево вносятся вершины, являющиеся потомками текущей вершины. Потомки берутся из главной таблицы.

   Существует четыре случая приделывания.

   (a) Если текущая вершина имеет тип "Слово"или "Константная строка". Не делается ничего - эта вершина не имеет потомков.

   (b) Если текущая вершина имеет тип Структура или Структура1. К текущей вершине приделывется потомки. Описание потомков текущей вершины лежит в массиве rWord начиная с индекса rStruct[z].r_word. Число потомков берется из Grammar.Trans[rStruct[z].i_struct].From.Word.j.

   (c) Если текущая вершина имеет тип Структура2. Текущей вершине тоже приделывется потомки. Описание потомков тоже лежит в массиве rWord начиная с индекса rStruct[z].r_word. но число потомков берется из словаря переводов.

   (d) Если текущая вершина имеет тип Выбор или Энум. Текущей вершине приделывется один потомок, и этот потомок - rWord[rStruct[z].r_word]

5.2 Построение Дерева фразы приемника

А теперь то, чего мы так желали - построение дерева-приемника, которое отображает структуру фразы на русском языке. Дерево строиться таким образом, что каждой вершине дерева-приемника (за исключением некоторых служебных слов) соответсвует вершина дерева источника. Это соответствие проставляется в Tree2[i].link.

1. Вносится первая вершина.

2. Производится обход дерева-приемника в прямом направлении, с достраиванием дерева в процессе обхода (в том же порядке, как это делалось при построении дерева источника). На каждом шаге в дерево вносятся вершины, являющиеся потомками текущей вершины.

   Существует пять случаев приделывания:

   (a) Если текущая вершина имеет тип «слово», или «Константная строка» то она не имеет потомков.

   (b) Если текущая вершина имеет тип «Структура» или «Структура1» приделываются в соответствии с трансляционной парой Grammar.Trans[i_struct]. При этом число потомков в дереве-источнике и дереве приемнике может не совпадать (за счет служебных слов в том и другом языке).

   (c) Если текущая вершина имеет тип «Выбор», то к текущей вершине должен добавиться потомок, соответстующий выбору в дереве источника. (Например подлежащее может быть существительным или местоимением. Если в дереве источнике подлежащее выражено местоимением, то и в приемнике оно должно быть выражено тем-же.) Например сказуемое может быть одним из 29 вариантов. Дерево источника в качестве потомка вершины "сказуемое"содержит одну вершину, которая соответсвует набору служебных слов в переводимой фразе - например "present perfect". Значит в дерево приемника должна быть внесена вершина соответсвующая этому самому "present perfect".

(d) Если текущая вершина имеет тип «Структура2». Это структура, внутренняя конструкция которй (вершины-потомки) определяется словарем. То есть формирование потомков такой вершины происходит так-же как в случае «структура», просто потомки берутся из другого места - из Slowo3.

(e) Если текущая вершина имеет тип «Энум». Энум - это тоже выбор, но выбор определяемый словарем. Так-же как в «Выборе» в качестве потомка вносится одна вершина, но тип ее определяется словарем Slowo3.

Примером вершины типа «Энум» является конструкция гр_сущ_п. гр_сущ_п - это некое понятие, имеющее смысл существительного, выраженное несколькими словами. В качестве возможных потомков эта вершина имеет гр_сущ_п1, гр_сущ_п2, гр_сущ_п3 и т.п. в зависимости от того сколькими словами выражено это выражение. Часто бывает, что количество слов в таком выражении по-английски и по-русски не совпадает. И тогда словарь определяет, что гр_сущ_п2 («main manager») переводится в гр_сущ_п4 («старшин начальник младшего дворника»)

5.3 Задание строковых значений

В принципе это можно было бы сделать в процессе построения дерева приемника. Но там и так черт ногу сломит, по этому задание строковых значений словам приемника я вынес в отдельную функцию.

Происходит рекурсивный обход дерева приемника слева направо.

Если вы знаете, что такое рекурсивный обход, этот абзац можете не читать. Функция обхода в качестве параметра имеет индекс текущей вершины дерева приемника. В каждой вершине эта функция производит некоторое действие, зависящее от типа вершины. Далее, если текущая вершина имеет потомков, то функция вызывает сама себя для каждого потомка. В каждой вершине действие повторяется - следует вызов потомков для потомков, и так пока все дерево не обойдется. Обход начинается с нулевой вершины.

В каждой вершине эта функция должна сделать следующее:

1. перевести слова, там где можно переводить слово в слово.

2. присвоить строковые значения там, где они берутся из lang.txt (например глагол[быть](...)), или из словаря выражений (например v1[быть])

3. передать строковые значения, в тех конструкциях, где есть составляющие, помеченные знаком «!» (например !глагол())

В какой вершине что делается, я здесь писать не буду, ибо неохота писать по-русски то, что уже написано на C++. Особо интересующиеся могут посмотреть исходник.

5.4 Вычисление параметров

Так вот прямо и хочется послать в соответсвующий раздел readme. Таки пошлю.

5.5 Формирование русской фразы

Изначально строка ответа является пустой. Происходит рекурсивный обход дерева приемника слева направо.(В том же порядке как это было при задании строковых значений.) Если вершина является словом, то это слово ставится в соответствующую форму, определяемую полем Tree[i].Form, и присоединяется к строке ответа.