YaYacc: Руководство программиста

DocumentId:GradSof-yayacc-r-PG-22.03.2001-1

May 7, 2001

`$Id: ProgrammingGuide_rus.tex,v 1.4 2001/05/05 19:35:27 rssh Exp $`

1 Введение

YaYacc – сокращение для Yet Another Yacc.

Программа создана как синтаксический анализатор, совместимый по алгоритму разбора и воспринимаемому языку с оригинальным `yacc` [1] (хороший учебник - [2]), но генерирующий код на C++ а не на C, как оригинальный yacc.

2 Общее описание работы с yayacc

1. Подготовка описания грамматики.
   - Программист составляет:
     - Заголовочный файл описания свойств генерируемого синтаксического анализатора (будем его называть `MyGrammarTag.h`), в котором определяется класс войств генератора синтаксического разбора, а именно:
       * Символьный тег грамматики – название класса, пусть к примеру `YYTag` В классе, должны быть определены следующие публичные поля:
         · YYType - тип данных, соответствующий терминальному символу – аналог YYTYPE в yacc.
         · YYInfoType - тип данных, для передачи информации основной программе во время разбора.
         · initStackSize - целая константа, определяющяя начальный размер стека разбора, аналог YYINITSTACKSIZE в yacc.
         · maxStackSize - целая константа, определяющяя максимально-возможный размер стека разбора, аналог YYMAXDEPTH в yacc.

         Пример :

```
struct MyGrammarTag
{
  typedef int YYType;
  const initStackSize = 100;
  const maxStackSize = 1000;
};
```

* Объявдение лексического анализатора, реализующего интерфейс Lexer<T>. Он должен либо наследоваться, либо специализироваться от класса Lexer<MyGrammarTag>, и содержать следующие методы:
  - `int readToken();` - должен прочесть следующую лексему и возвратить ее код.
  - `typename MyGrammarTag::YYType yylval() cosnt` - должен возвратить последний терминальный символ, соответствующий этой лексеме.

  Пример:

```
tempate<>
class Lexer<MyGrammarTag>
{
 private:
   istream& in_();
 public:
    Lexer(istream& in)
     :in_(in) {}
    ~Lexer() {}

    int readToken();
    MyGrammarTag::YYTag getLastValue() const;
};
```

- Файл описания грамматики и действий синтаксического разбора, совместимый с оригинальным yacc. (будет его наывать Grammar.y). Вы должны включить файл `MyGrammarTag.h` в первой секции кода `Grammar.y`.
- Файл определеня констант из `MyGrammarTag.h` - будем его называть `MyGramarTag.cpp`

2. Генерация кода
   - Разработчик запускает yayacc с ключом `-n MyGrammarTag` (подробнеебсм. в разделе опции)
   - Генерируется файл (по умолчанию: y.tab.cpp (а также, возможно, y.tab.h))
3. Использование кода:

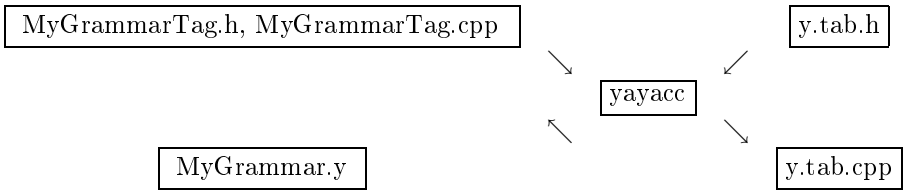
- Разработчик использует класс Parser<MyGrammarTag>, имеющий следующую public спецификацию:

```
template<>
class YYParser<MyGrammarTag>
{
 ....
 public:

   YYParser(YYLexer<MyGrammarTag>& lexer, typename MyGrammarTag::YYInfoType);
  ~YYParser();

   int parse();
   typename MyGrammarTag::YYType  getValue() const;

   typename MyGrammarTag::YYInfoType getInfo() const;

   void setDebugLevel(int debugLevel);

};
```

  Этот класс определен в генерируемом заголовочном файле `<y.tab.h>`

  В генерируемом файле находится код метода класса `Parser<MyGrammarTag>` parse.

Таким образом, общая ситуация может быть выражена следующей схемой:



3 Подробное рассмотрение

Теперь перейдем к более подробному рассмотрению процесса работы yayacc. Итак, вначале – параметры граммматик.

3.1 Параметры свойств грамматик: на что они влияют

subsectionYYType

Тип терминалов грамматики. Семантика у него такая-же, ка и YYTYPE в стандартном yacc: т. е. MyGrammarTag::YYType используется в том месте, где традиционно стоял YYTYPE; YYType должен обладать следующими свойствами:

- Быть конструируемым, т. е. обладать конструктором по умолчанию.
- Быть присваиваемым, т. е. обладать оператором присваивания и конструктором копии.

Если в вашем понимании терминальный символ это более сложная структура, для которой использование глубокого копирования неэффективно (например, узел синтаксического дерева), то в качестве YYType лучше использовать указатели на эту структуру. В таком случае, не забудьте вставить цикл удаления элементов в ваш код восстановления от ошибок.

subsubsectionПредложение `%union`

Если вы используете `%union` клаузу в файле `Grammar.y`, то соответствующее определение типа генерируется автоматически, как конкатенация строк `MyGrammarTag` и `YYType`. Так, например, если вы запускаете yyac с опцией `-n Calculator`, и в вашем `Calculator.y` файле есть определиние union

```
%union {
  int i;
  char ch;
};
```

то в результате работы в `y.tab.h` и в `y.h.cpp` появятся следующие строки:

```
typedef union {
    int i;
    char ch;
}  CalculatorYYTag;
```

В `Calculator.h` вы можете либо продублировать это определение, либо включить `y.tab.h` и нписать что-то вроде:

```
struct Calculator
{
 typedef CalculatorYYType YYType;
 ...
};
```

subsectionYYInfoType

Этот тип данных, использующийся для передачи информации между синтаксическим анализатором и программным окружением из входов анализатора.

Более подробно: Классу Parser во время создания передается ссылка на объект типа YYInfoType. Вы можете пользоваться ей, для передачи дополнительной информации из/в синтаксичесуого анализатора

subsectionmaxStackSize

Это должна быть целая константа, определяющая максимальный размер стека синтаксического анализатора. Ее можно считать, как верхнюю границу для максимально-возможного количества нередуцируемых символов. Значение по умолчанию в `byacc` - 10000.

Кстати, вопрос: зачем вобще нужно ограничивать сверху размер стека – что-бы при использовании синтаксического анализатора в сетевых сервисах не было возможности для атак типа DOS.

subsectioninitStackSize

Эта должна быть целая константа, определяющая начальный размер стека анализатора. Ее можно считать как верхнюю границу для количества нередуцируемых символов в текузей программе. Для грамматики калькулятора это будет что-то около пяти. Значение этой константы в `byacc` по умолчанию – 100.

subsectionПример: описание тега грамматики

```
struct MyGrammarTag
{
 typedef int YYType;
 typedef bool YYInfoType;
 static  const int maxTableSize;
 static  const int initTableSize;
}
```

3.2 Интерфейс с лексическим анализатором

Лексический анализатор передается[1] объекту YYParser при его создании. Шаблон для лексического анализатора описан в интерфейсном файле `GradSoft/YaYacc.h` и имеет вид:

```
template<class YYTag>
class YYLexer
{
public:
  int      readToken();
  typename YYTag::YYType  getLastValue();

  YYLexer() {};
  virtual ~YYLexer() {};

private:

  YYLexer(const YYLexer&);
  YYLexer& operator=(const YYLexer&);

};
```

Вы должны специализировать этот шаблон и /или/ наследовать свой лексический анализатор от него.

Важными там являются две функции:

[1] по ссылке

- `int readToken()` – считать следующую лексему, сохранить соответствующее значение во внутренней переменной состояния. (если она есть).
- `getLastValue()` – возвратить значение последней прочитанной лексемы.

Пример лексического анализатора находится в файлах `CalculatorLexer.h` (соответственно CalculatorLexer.cpp) в поддиректории `demo/calculator` дистрибутива YaYacc

3.3 Интерфейс с основной программой

- Вызов метода YYParser<MyGrammarTag>::parse - полный аналог вызова yyparse() в yacc.
- Кроме этого ваша программа может обмениваться информацией с синтаксическим анализатором посредством переменной `yyinfo`. Она имеет тип `typename MyGrammarTag::YYInfoType`. Вы передаете ссылку на эту переменную в конструкторе синтаксического анализатора, в правилах вы модете присваивать yyinfo значения; после сеанса разбора вы можете получить значение yyinfo с помощью метода `YYParser<MyGrammarTag>::getYYInfo()`;
- Так-же, как и в традиционном yacc вы можете определить функцию обработки ошибок `yyerror`. Ее определение должно находится в файле `MyGrammar.y` и выглядеть следующим образом:

  ```
  template<>
  void YYParse<MayGrammarTag>::yyerror(const char* msg)
  {
   .. you cleanup here
  }
  ```

  В случае сложной структуры `YYType` вам возможно будет необходимо сделать очистку стека: в таком случае имейте в виду, что у вас в стеке разбора есть (yyssp - yyss) элементов и их нумерация начинается с нуля.

  ```
  template<>
  void YYParse<MayGrammarTag>::yyerror(const char* msg)
  {
   for(inti i=0; i<yyn; ++i) {
     delete yyvs[i];
   }
  }
  ```

- Вы можете получить сообщение о последней ошибке с помощью метода `YYParser<YYTag>::getErrorMessage()`. Если ошибки не произошло, то эта функция возращает NULL.

- Отладочная печать может быть встроенна в синтаксический анализатор. Для этого должен быть определен препроцессорный символ YYDEBUG. В отличие от традиционного уасс, отладочный вывод включается не с помошью переменной среды окружения, а из программы, с помошью метода `YYParser<MyGrammarTag>::setDebugLevel` . Для включения отладочного вывова вы должны вызвать этот метод с параметром больше нуля.

4 Пример: простой калькулятор

Находится в директории `demo/calculator/` в дистрибутиве уауасс.

5 Опции командной строки

- `-n <MyGrammarTag>` - эта добавленная нами опция, которая определяет значение символа `YYTag`.
- `-b <file-prefix>` - традиционная опция, изменяет значения префикса генерируемых файлов.
- `-d` - традиционная опция, вызывает генерацию файла `y.tab.h`. В нашем варианте эта опция включена по умолчанию.
- `-l` - традиционная опция, отключить генерацию `#line` директив препроцессора.
- `-o <file-name>` - традиционная опция, генерировать файл `file-name` вместо `y.tab.cpp`.
- `-p <symbol-prefix>` - традиционная опция, использовать `symbol-prefix` вместо `yy`. Мы не гаранитруем совместимость уауасс с этой опцией.
- `-r` - традиционная опция, генерировать отдельно файл таблиц `y.tab.cpp` и файл кода `y.code.cpp`
- `-t` - традиционная опция - включать генерирование отладочного вывода в генерируемый код.
- `-v` - традиционная опция - гененрировать читаемое описание грамматики в файле `y.output`.

6 Распространение YaYacc runtime в ваших пакетах

Просто перенесите содержимое подкаталога `interfaces` в какой-то подкаталог вашего проекта и укажите его как еще один источник включаемых файлов в опциях C++ компилятора.

Если ваш проект инсталлирует включаемые файлы, которые каким-то образом используют уауасс интерфейсы, то при инсталляции создайте в `<prefix>/include` [2] поддиректорию GradSoft и инсталлируйте туда файл `YaYacc.h`

7 Перечень изменений

1. 07.05.2001 - добавлено описание метода getErrorMessage.
2. 26.04.2001 - опубликована первая редакция.
3. 22.03.2001 - создан.

References

[1] Stephen C. Johnson. *yacc - Yet Another Compiler Compiler*. bell-labs, 1978. aviable online at http://cm.bell-labs.com/7thEdMan/vol2/yacc.bun.

[2] John R. Levine Tony Mason and Doug Brown. *Lex & Yacc*. O'Reilly and Associates, 1992. ISBN 1-56592-000-7.

[2] здесь prefix - место инсталляции Вашего пакета