YaYacc: Programmers Guide

DocumentId:GradSof-yayacc-e-PG-27.03.2001-1

June 7, 2001

`$Id: ProgrammingGuide_eng.tex,v 1.6 2001/05/21 10:10:24 vaa Exp $`

1 Introduction

YaYacc – abridgment for Yet Another Yacc.

The program was created as a syntactical analyzer, which is compatible with original `yacc`| citeyacc ([1] - good tutorial) on algorithm analysis and perceived language `LALR(1)`, and it generates the code by C++ language but not by C language as original yacc.

2 General description of work with yayacc

1. Default of grammar's description.
   - The programmer composes:
     - Header file which describes the property of generated syntactical analyzer (we will call it `YYProperties.h`), in which class of generator's properties syntactical analysis must be defined. The properties are:
       * Symbol tag of grammar – the name of class, as a example `YYTag`
         The following public fields must be defined in the class:
         · YYType - type of data, which corresponds to terminal symbol – the analogue YYTYPE in yacc.
         · initStackSize - the interger constant, which defines the first size of stack's analysis, analogue of YYINITSTACK-SIZE in yacc.
         · maxStackSize - the integer constant, which defines as much as possible the size of stack's analysis, analogue of YY-MAXDEPTH in yacc.

         The example:

```
struct MyGrammarTag
{
  typedef int YYType;
```

```
  const initStackSize = 100;
  const maxStackSize = 1000;
};
```

The following conditions must be fulfil for correct work of generated syntactical analyzer: YYType is the class with constructor by default, which can copy and bind itself (i.e. Copyable, Assignable and DefaultConstructable)

* The notice of lexical analyzer realizes interface Lexer<T>. It must inherit itself or specialize from the class `Lexer<MyGrammarTag>`, and contain the following methods:
  - `int readToken();` - must read next lexeme and return its code.
  - `typename MyGrammarTag::YYType yylval() cosnt` - must return the last terminal symbol, corresponding this lexeme.

  Example:

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

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

- The file of description of grammar and actions syntactical analysis, compatible with the original yacc. (we will call it Grammar.y)

2. The generation of code
   - Developer start up yayacc with the key `-n MyGrammarTag` (in detail look to a part "options")
   - The file generates(on default: y.tab.cpp (and may be y.tab.h))
3. Using the code:
   - Application programmer uses the class Parser<MyGrammarTag>, which has following public specification:

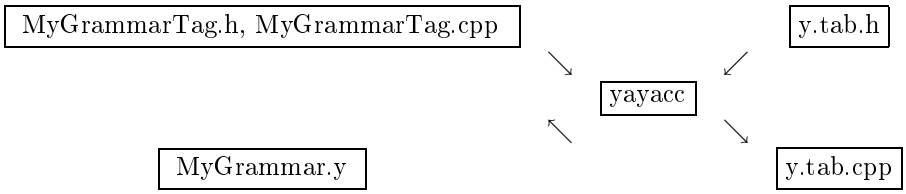
```
template<>
class YYParser<MyGrammarTag>
{
 ....
 public:
```

```
    YYParser(YYLexer<MyGrammarTag>& lexer);
   ~YYParser();

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

    void setDebugLevel(int debugLevel);

};
```

This class is defined in generated title file `<y.tab.h>`

The code of class's method is situated in generated file `Parser<MyGrammarTag>` parse.

Thus, general situation can be expresses the following scheme:



3 Detail examination

Let's go to detail examination of process working yayacc. So, at beginning – parameters of grammar.

3.1 grammar properties: on which are they influence

subsectionYYType

Terminal grammar type. It has semantics the same as YYTYPE in standard yacc has: i.e. MyGrammarTag::YYType is used in that place, where traditionally YYTYPE situates; YYType must obtain the following properties:

- To be constructible, i.e. to obtain the constructor by default.
- To be assignable, i.e. possess the operator of assignment and constructor copy. nent

If in your comprehension the terminal symbol is the more difficult structure, for which using deep copy is not effective (for example, knot of the syntactical tree), then better to use pointers on this structure in capacity of YYType. In this case, don't forget to put the deleting elements cycle in your code renewal from mistakes.

subsubsection%union statement

If you use %union statement in file Grammar.y, then corresponding definition of type is generated automatically as concatenation of lines MyGrammarTag and YYType. Thus, for example, if you run yyac with option -n Calculator, and there is the definition of union in your Calculator.y file.

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

then in work result the following lines will appear in y.tab.h and in y.h.cpp :

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

In Calculator.h you may either duplicate this definition or include y.tab.h and write something like that:

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

subsectionYYInfoType

This data type, which is used for information passing between syntactical analyzer and program environment from enters analyzer.

More detail: At the time of creation the reference is passed into the class Parser on the object as type YYInfoType. You may use it for passing additional information from/into syntactical analyzer.

subsectionmaxStackSize

It must be integer constant, which define maximal size of stack of syntactical analyzer. It may be considered as top limit for maximal possible quantity of no reducible symbols. The value on default in yacc - 10000.

By the way, it is question: for what is it necessary to limit from the top the size of stack – for when use the syntactical analyzer in network services it isn't any possibility for attack as type DOS.

subsectioninitStackSize

This must be the integer constant, which define the initial size of the analyzer stack. It may be considered the top limit for quantity of no reducible symbols in current program. For grammar calculator it will be near five. Constant value in yacc on preterition – 100.

subsectionThe example: describing of the grammar tag

```
struct MyGrammarTag
```

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

3.2 The interface with lexical analyzer

A lexical analyzer is passed [1] to the object YYParser when its creation.

Template for lexical analyzer is described in interface file `GradSoft/YaYacc.h` and has a appearance:

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

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

private:

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

};
```

You must assign a specialization this template and /or/ inherit your lexical analyzer from it.

Two following functions are important:

- `int readToken()` – read next lexeme, keep correspond value in inside status variable. (if it exists).
- `getLastValue()` – return the value of last read lexeme.

The example of lexical analyzer is situated in files `CalculatorLexer.h` (corresponding CalculatorLexer.cpp) in subfolder `demo/calculator` of distribute YaYacc

3.3 The interface with the main program

- The call of method YYParser<MyGrammarTag>::parse is the analogue of the call yyparse() in yacc.

[1] on reference

- Besides, your program can change of information with syntactical analyzer by means of variable `yyinfo`. It has type `typename MyGrammarTag::YYInfoType`. You pass the reference on this variable in constructor of syntactical analyzer, in rules you can appropriate yyinfo meanings; after seance you can take meaning yyinfo with the help of method `YYParser<MyGrammarTag>::getYYInfo()`;

- Like in traditional yacc you can define the function of processing mistakes `yyerror`. Its definition must be situated in file `MyGrammar.y` and look as following:

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

  In the case of complicated structures `YYType` you will necessary do cleaning stack: In this case have in view that there are elements from 0 to (yysspyyss) in your examination.

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

- Also, you can get text of last error message with help of method `YYParse<YYTag>::getErrorMessage`. Note, that this method returns NULL, if no errors was occur.

- Debug printing can be built-in syntactical analyzer. Preprocessor symbol YYDEBUG must be defined for this. As against traditional yacc, checkout conclusion is included not with the help of environment variable, but from the program with the help of method `YYParser<MyGrammarTag>::setDebugLevel`. For enabling debug printing you must call this method with parameter, which is bigger then 0.

4 The example: simple calculator

It is situated in directory `demo/calculator/` in yayacc distributive.

5 Command line options

- `-n <MyGrammarTag>` - this is added by us option, which defines the value of the `YYTag` symbol.

- `-b <file-prefix>` - the traditional option, it changes value of prefix of generated files.
- `-d` - the traditional option, it calls the file generation. `y.tab.h`. In our variant this option is included on default.
- `-l` - the traditional option, to disable the generation of `#line` preprocessor directives.
- `-o <file-name>` - the traditional option, to generate the file `file-name` in place of `y.tab.cpp`.
- `-p <symbol-prefix>` - the traditional option, to use `symbol-prefix` in place of `yy`. We don't guarantee the compatibility of yayacc with this option.
- `-r` - the traditional option, to generate separately the file of tables `y.tab.cpp` and file of code `y.code.cpp`
- `-t` - the traditional option - to include generation of debug printing in generated code.
- `-v` - the traditional option - to generate the readable description of grammar parser in file `y.output`.

6 The distribution of YaYacc runtime in your pack

Simple transport the contents of subcatalogue `interfaces` in some subcatalogue of your project and set it as one more source of included files in options of your C++ compiler.

If your project installs included files, which use yayacc interfaces, then during installation create in `<prefix>/include` [2] subdirectory GradSoft and install there the file `YaYacc.h`

7 The list of changes

1. 07.05.2001 - added description of getErrorMessage method.
2. 05.05.2001 - Bibliography added.
3. 28.04.2001 - English version was created.

[2] here prefix - installation place of your pack

References

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