

熟悉 **Parser Generator**

潘宇

September 2019

目录

1 Parser Generator 的安装与使用	3
1.1 Parser Generator 的安装	3
1.2 Parser Generator 的使用	3
1.2.1 创建 parser generator 项目。	5
1.2.2 编写 Lex、Yacc 程序。	7
1.2.3 将 Lex、Yacc 程序编译为 C、C++ 程序。	7
1.2.4 创建 VS 工程项目, 将 parser generator 2 生成的头文 件和源文件添加到项目中。	8
1.2.5 设置工程环境, 编译运行。	9
1.2.6 PG 往年使用经验	14
2 作业要求	14
3 作业解析	15
3.1 作业目的	15
3.2 作业思路	15
3.3 讲义中 Yacc 程序讲解	15
3.4 Yacc 程序修改——简单表达式计算	17
3.5 中缀表达式转后缀表达式	20
3.6 思考——符号表	22
3.6.1 思考题要求	22
3.6.2 思考题提示	22

1 Parser Generator 的安装与使用

1.1 Parser Generator 的安装

Parser Generator 的安装程序位于开发工具文件夹下的安装程序文件夹下，其中 ParGen.exe 为 Parser Generator 的安装程序，直接运行安装即可。

1.2 Parser Generator 的使用

在详细说明之前，先简单介绍 Parser Generator 2 的使用流程主要分为以下几步：

- a 创建 parser generator 项目。
- b 编写 Lex、Yacc 程序
- c 将 Lex、Yacc 程序编译为 C、C++ 或 java 程序
- d 创建 VS 工程项目，将 parser generator 2 生成的头文件和源文件添加到项目中
- e 设置工程环境，编译运行

其中前四步较为简单，最后一步较为复杂且根据 VS 版本, 32/64 位, C/C++ 语言, Debug/Release 等情况的不同而方法不唯一，因此同学们需要认真处理。

王刚老师使用 Win10 专业版 Version 1067, Visual Studio 2015 的机器系统已尝试了各种组合，均成功，具体如下：

语言	调试?	位数	库目录	运行库设置	附加依赖项	其他设置
C	Debug	32	msvc2015	多线程调试 (/MTd)	ylmtd.lib	预处理器定义: YYDEBUG
C	Debug	64	msvc64	多线程调试 (/MTd)	ylmtd.lib	预处理器定义: YYDEBUG
C	Release	32	msvc2015	多线程(/MT)	ylmt.lib	
C	Release	64	msvc64	多线程(/MT)	ylmt.lib	
C++	Debug	32	msvc2015	多线程调试 (/MTd)	ylmtd.lib	预处理器定义: YYDEBUG
C++	Debug	64	msvc64	多线程调试 (/MTd)	ylmtd.lib	预处理器定义: YYDEBUG
C++	Release	32	msvc2015	多线程(/MT)	ylmt.lib	
C++	Release	64	msvc64	多线程(/MT)	ylmt.lib	
C	Debug	32	msvc32	多线程调试 (/MTd)	ylmtd.lib;legacy_stdio_definitio ns.lib; 后者是提供一些过时函数	预处理器定义: YYDEBUG 链接器-命令行-其他选项: /FORCE 后者解决 PG 单线程和 VC 多线程的重复定义
C	Release	32	msvc32	多线程(/MT)	ylmtd.lib;legacy_stdio_definitio ns.lib;	链接器-命令行-其他选项: /SAFESEH:NO

VS2015 下使用 PG 自带的库也能成功, 见表格最后两行。但只能是 C 程序, C++ 程序由于 VC 版本的问题, 无法使用 PG 自带库编译运行。也已有同学在 VS 2013、2012 上用 PG 自带的库编译成功, 也是 C 程序可以, C++ 程序不可以, 这一点大家注意。

其中 msvc32 为 PG 自带的库, msvc2015 和 msvc64 我们已提供, 位于开发工具文件夹下的 Parser Generator 编译库文件夹中, 将 msvc2015 和 msvc64 文件夹放置于 *DIR\CPP\Lib* (其中 *DIR* 为你安装 PG 的路径, 默认路径为 *C:\Program Files(x86)\Parser Generator 2*) 中, 一般均可成功使用 (即在你的机器上这两个编译好的库可以编译运行 Yacc 和 Lex 程序), 如不成功, 可参考开发工具文件夹下的 PG 库编译方法.docx 中的方法自己编译库, 其中所需要的 lbs 文件和 dll 动态链接库同样位于开发工具文件夹下的 Parser Generator 编译库文件夹中。

以上所说的整体五个步骤的大致流程可参看李鹏师兄的 CSDN 博客:

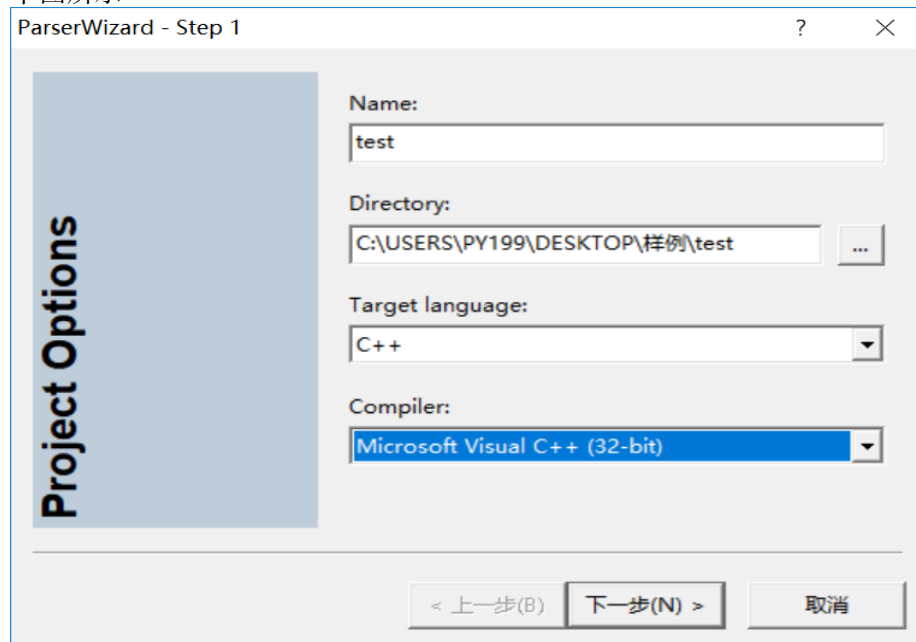
<http://blog.csdn.net/lpstudy/article/details/51330063>. 但需要注意该方案不支持 C++ 语言的文件流输入输出, 因此我这里给出一种用于 C++ 语言, VS2015/2017, 32 位, Release 模式下 (即表格第七行) 支持 c++ 语言的文件流读入写出方案作为例子进行 Parser generator 2 使用的讲解, 打算选用其他模式的各位同学可举一反三进行配置。

1.2.1 创建 parser generator 项目。

打开 Parser generator 2, 选 Project/ParserWizard 菜单项, 开始项目的创建。

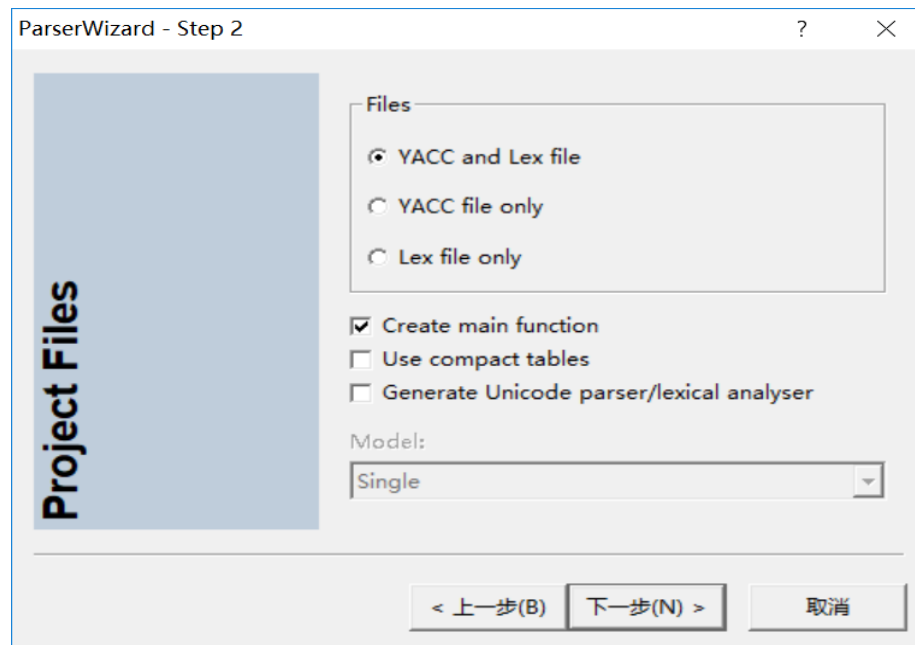
第一步: 确定项目名称、保存目录、生成分析程序语言 (C、C++...)、开发环境 (VC, BC...).

这里命名为 test, 选择目标语言为 C++, 编译器为 vc++(32-bit), 如下图所示:



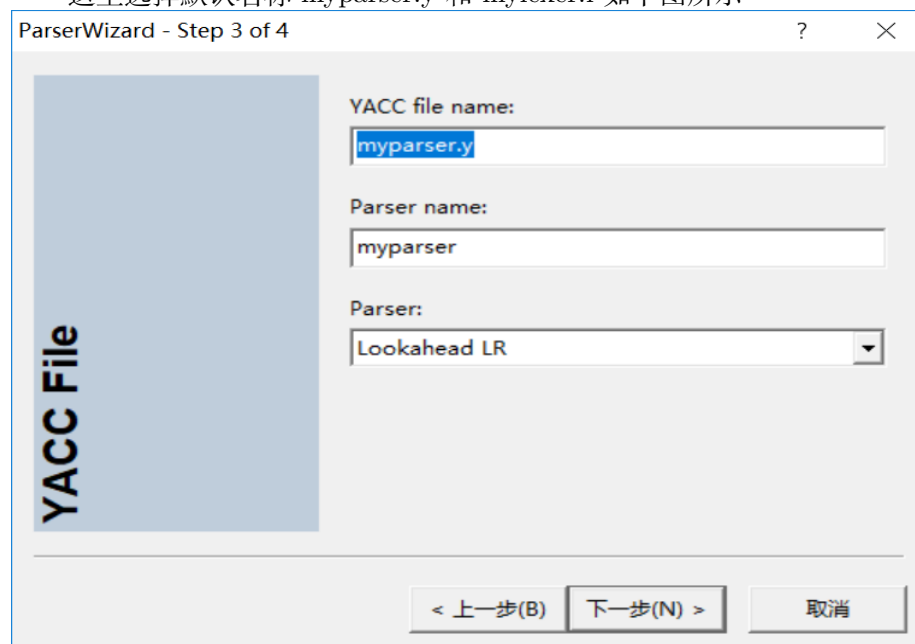
第二步: 决定是否生成 Lex 和 (或) Yacc 文件? 是否创建 main 函数? 是否使用压缩的 LR 分析表? 是否生成 Unicode 程序? single/multiple 模式?

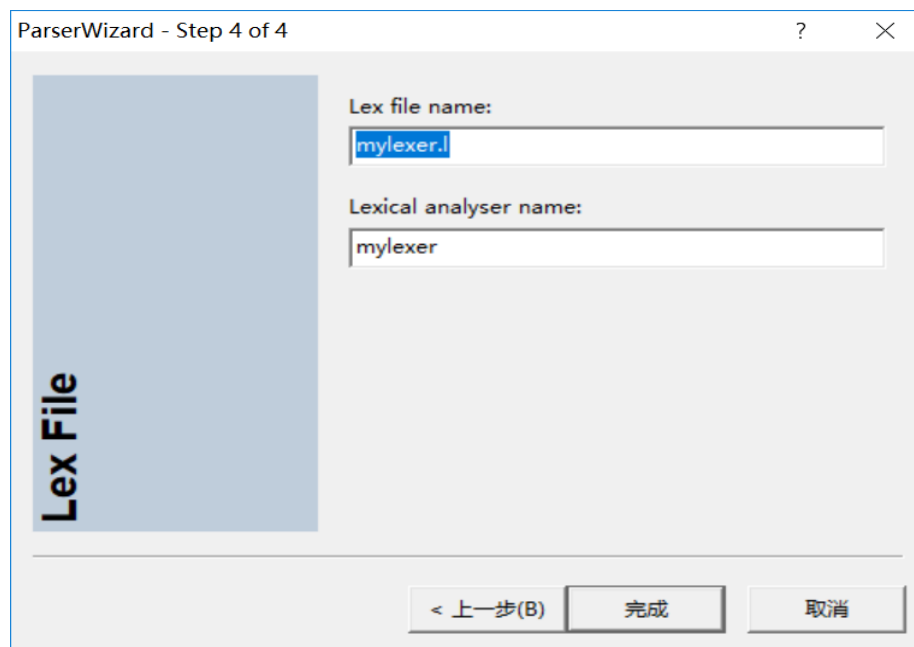
这里同时测试 Lex 和 Yacc 文件, 创建 main 函数, 如下图所示:



第三步、第四步：设置 lex、yacc 文件名——两者前缀不能相同！否则生成同名 C 文件，会产生冲突。

这里选择默认名称 myparser.y 和 mylexer.l 如下图所示：





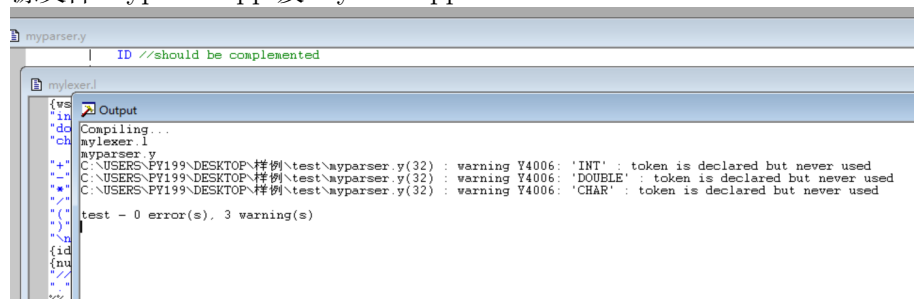
点击完成，完成项目的创建。










1.2.2 编写 Lex、Yacc 程序。

这里使用李鹏师兄的测试代码进行测试，代码已保存在开发工具文件夹下的 test 文件夹中的 myparser.y 和 mylexer.l 中，复制粘贴内容即可。

1.2.3 将 Lex、Yacc 程序编译为 C、C++ 程序。

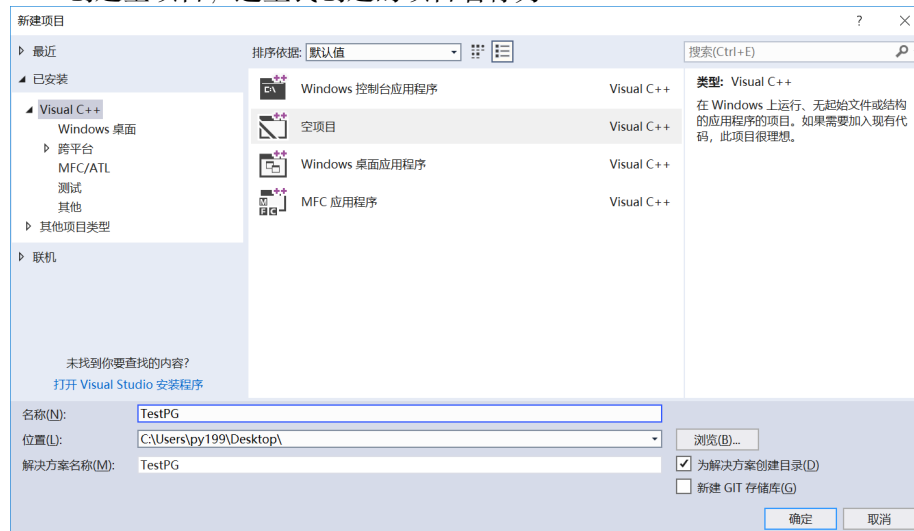
选 Project/build 菜单项, 生成对应头文件 myparser.h 及 mylexer.h 和源文件 myparser.cpp 及 mylexer.cpp



	mylexer.cpp	2019/9/1 23:20	C++ source file	9 KB
	myparser.cpp	2019/9/1 23:20	C++ source file	17 KB
	mylexer.l	2019/9/1 22:54	Dictionary File	2 KB
	mylexer.h	2019/9/1 23:20	Header file	2 KB
	myparser.h	2019/9/1 23:20	Header file	3 KB
	test.pgp	2019/9/1 22:53	Parser Generator Pr...	1 KB
	mylexer.v	2019/9/1 23:20	Verbose File	6 KB
	myparser.v	2019/9/1 23:20	Verbose File	5 KB
	myparser.y	2019/9/1 22:55	YACC Source File	2 KB

1.2.4 创建 VS 工程项目，将 parser generator 2 生成的头文件和源文件添加到项目中。

创建空项目，这里我创建的项目名称为 TestPG.

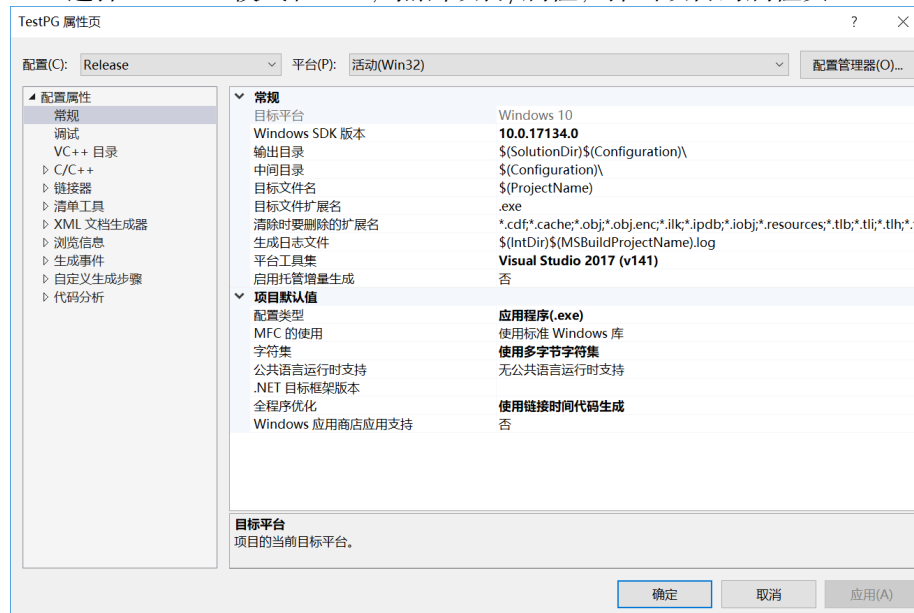


点击头文件/添加/现有项将 myparser.h 及 mylexer.h 添加入头文件, 点击源文件/添加/现有项将 myparser.cpp 及 mylexer.cpp 添加入源文件。如下图所示:



1.2.5 设置工程环境，编译运行。

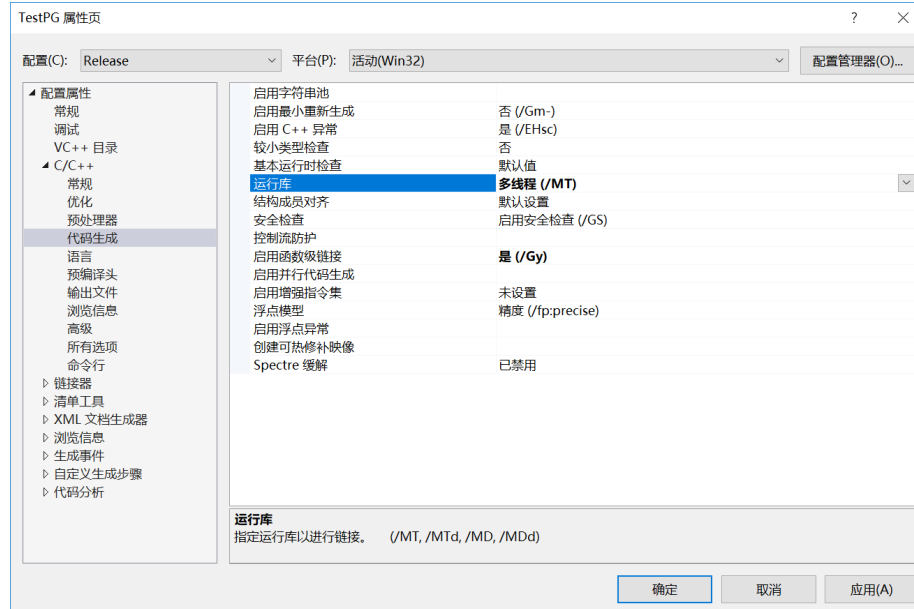
选择 Release 模式和 x86，点击项目/属性，弹出项目的属性页



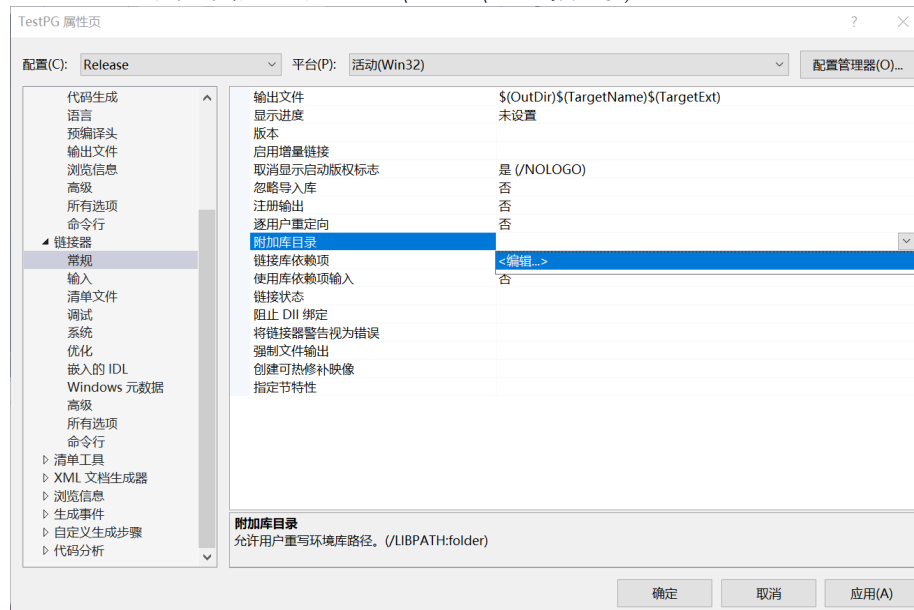
点选 C/C++ 的常规页下的附加包含目录，添加 `DIR\CPP\Include` (与之前说明相同，其中 DIR 为你安装 Parser Generator 2 的路径，默认路径为 `C:\Program Files (x86)\Parser Generator 2`)

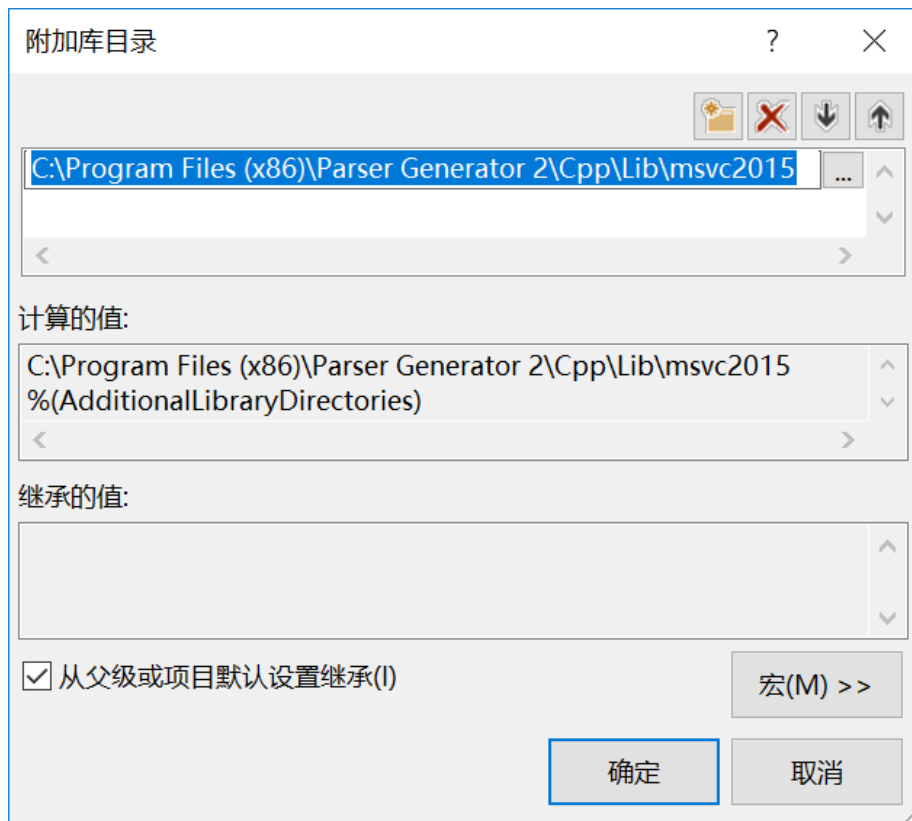


點選 C/C++ 的代码生成页，运行库选项修改为多线程 (/MT)

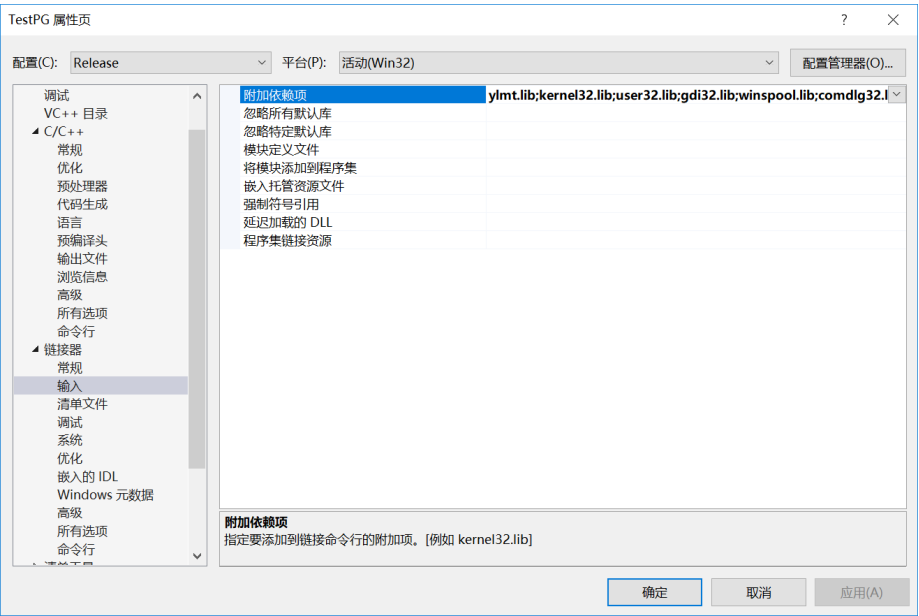


點選链接器的常规页的附加库目录，添加 `DIR\CPP\Lib\msvc2015` (与之前说明相同，其中 DIR 为你安装 Parser Generator 2 的路径，默认路径为 `C:\Program Files (x86)\Parser Generator 2`，注意不要忘记将 `msvc2015` 文件夹放置于 `DIR\CPP\Lib` 路径中)

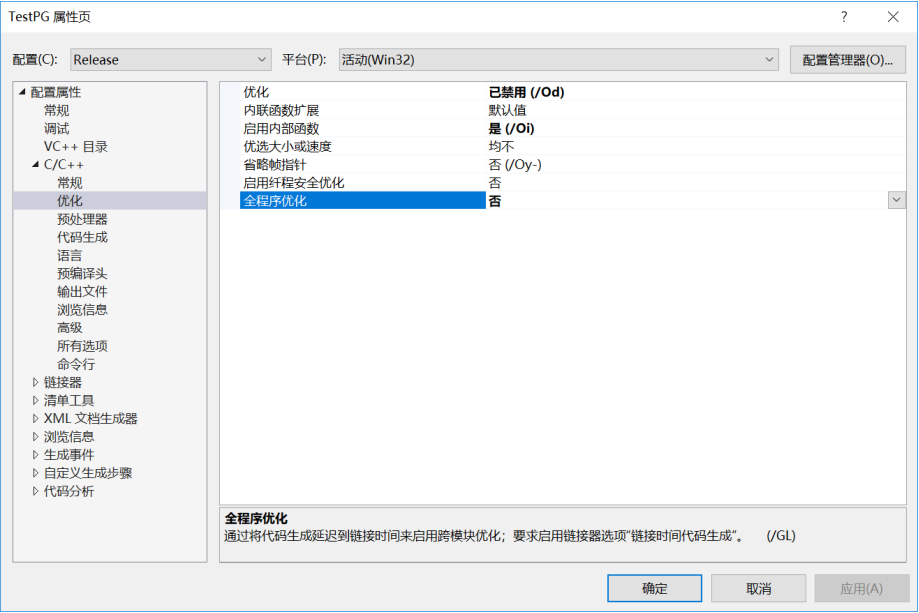




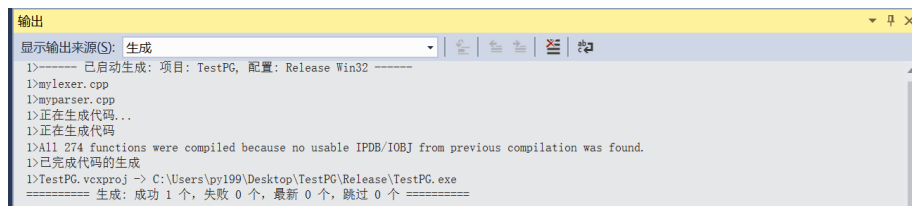
点击链接器的输入页，添加 ylmt.lib:



若遇到 Error：对象或库文件 release\...lib 是使用比创建其他对象所用编译器旧的编译器创建的：请重新生成旧的对象和库，点击 C/C++ 的优化页将全程序优化关闭即可。

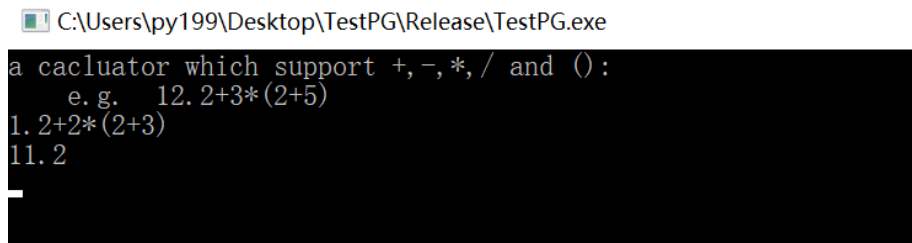


全部完成后生成解决方案：



```
输出
显示输出来源(S): 生成
1>----- 已启动生成: 项目: TestPG, 配置: Release Win32 -----
1>mylexer.cpp
1>myparser.cpp
1>正在生成代码...
1>正在生成代码
1>All 274 functions were compiled because no usable IPDB/IOBJ from previous compilation was found.
1>已完成代码的生成
1>TestPG.vcxproj -> C:\Users\py199\Desktop\TestPG\Release\TestPG.exe
===== 生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个 =====
```

成功后 ctrl+F5 运行即可, 结果成功演示如下:



```
C:\Users\py199\Desktop\TestPG\Release\TestPG.exe
a caculator which support +, -, *, / and ():
e. g. 12. 2+3*(2+5)
1. 2+2*(2+3)
11.2
```

1.2.6 PG 往年使用经验

a Parser Generator 注意事项 (王艺霖):

注意代码生成-> 运行库与 Parser Generator 的 lib 一致。对于 vs2015, 如果使用 msvc32 的 lib, 则必须使用 ylmtrid.lib 和 ylmtri.lib, 而且必须使用 C++ 语言, 运行时可能会提示找不到 vc6 运行库, 从网上下载即可。对于更低版本的编译器, 有可能与 vs2015 不兼容, 这时有两个选择, 自己编译 lib, 或者使用 msvc32 的 lib 并且使用 C++ 语言。

b 建议第一个 yacc 的.y 就选 c 32 位, 先别拷贝老师的代码, 生成.h.c 文件, 然后去 vs 里配置, 看看能不能运行, 能运行就是.y 不对, 否则就是 vs 配置的问题, 这样比较快 (冯永琦)

2 作业要求

安装 Parser Generator 软件, 熟悉其使用, 对讲义中简单表达式计算的 Yacc 程序进行修改。

1. 将所有的词法分析功能均放在 yygettoken 函数内实现, 为 +、-、*、\、(、) 每个运算符及整数分别定义一个单词类别, 在 yygettoken 内实现代码, 能识别这些单词, 并将单词类别返回给词法分析程序。

2. 实现功能更强的词法分析程序，可识别并忽略空格、制表符、回车等空白符，能识别多位十进制整数。
3. 修改 Yacc 程序，不进行表达式的计算，而是实现中缀表达式到后缀表达式的转换。

3 作业解析

3.1 作业目的

通过本次实验，希望同学们能够熟悉并掌握 Parser Generator 的使用。结合课上所学知识增进对词法分析过程的了解，学会 Yacc 程序的编写，巩固所学知识，提高实践能力。

3.2 作业思路

根据作业要求，可以看出前两条作业要求返回简单表达式计算的值，而第三条作业要求实现中缀表达式到后缀表达式的转换，因此返回的是字符串类型的表达式。因此建议同学们分写两个程序。讲义中简单表达式计算的 Yacc 程序位于第二章 PPT 的 55-60 页。第一条作业要求的修改重点在于定义单词类别去替代运算符及整数，将字符流转化为一个一个的 token，第二条作业要求的修改可参考第二章 PPT 的第 95-96 页。第三条作业要求中缀转后缀，重点在于修改翻译模式和返回值，类比识别多位十进制整数完成实现识别标识符，并添加有关标识符的上下文无关文法。

3.3 讲义中 Yacc 程序讲解

讲义中有关简单表达式计算的 Yacc 程序如下：

```
1  %{
2  /*****
3  expr.y
4  ParserWizard generated YACC file.
5
6  Date: 2016年10月18日
7  *****/
8  #include <iostream>
9  #include <cctype>
10 using namespace std;
11 %}
12
13 //////////////////////////////////////
14 // declarations section
```

```

15 %include {
16 #ifndef YYSIYPE
17 #define YYSIYPE double
18 #endif
19 }
20
21 // parser name
22 %name expr
23
24 // class definition
25 {
26     // place any extra class members here
27     virtual int yygettoken();
28 }
29
30 // constructor
31 {
32     // place any extra initialisation code here
33 }
34
35 // destructor
36 {
37     // place any extra cleanup code here
38 }
39
40 // place any declarations here
41 %left '+' '-'
42 %left '*' '/'
43 %right UMINUS
44
45 %%
46
47 ///////////////////////////////////////////////////////////////////
48 // rules section
49
50 // place your YACC rules here (there must be at least one)
51
52 lines :      lines expr '\n' { cout << $2 << endl; }
53       |      lines '\n'
54       ;
55
56
57 expr :      expr '+' expr { $$ = $1 + $3; }
58       |      expr '-' expr { $$ = $1 - $3; }
59       |      expr '*' expr { $$ = $1 * $3; }
60       |      expr '/' expr { $$ = $1 / $3; }
61       |      '(' expr ')' { $$ = $2; }
62       |      '-' expr %prec UMINUS { $$ = -$2; }
63       |      NUMBER
64       ;
65
66 NUMBER :    '0' { $$ = 0.0; }
67          |    '1' { $$ = 1.0; }
68          |    '2' { $$ = 2.0; }
69          |    '3' { $$ = 3.0; }
70          |    '4' { $$ = 4.0; }
71          |    '5' { $$ = 5.0; }
72          |    '6' { $$ = 6.0; }
73          |    '7' { $$ = 7.0; }
74          |    '8' { $$ = 8.0; }
75          |    '9' { $$ = 9.0; }
76          ;
77

```



```

78 %%
79
80 //////////////////////////////////////////////////
81 // programs section
82
83 int YYPARSENAME::yygettoken()
84 {
85     // place your token retrieving code here
86     return getchar();
87 }
88
89 int main(void)
90 {
91     int n = 1;
92     expr parser;
93     if (parser.ycreate()) {
94         n = parser.yyparse();
95     }
96     return n;
97 }
98

```

其中 1-11 行用于添加所需的头文件和命名空间，注释部分可添加描述 yacc 文件信息。declarations section 部分的 YYSTYPE 用于确定 \$\$ 变量类型，这里由于需要返回的是简单表达式计算结果，因此声明为 double 类型。第 41-43 行用于声明运算符的左右结合性及优先级，按照优先级由低到高的顺序声明，同一级的运算符放在同一行。rules section 部分为语法分析部分，包括上下文无关文法及翻译模式。大括号内的部分为执行动作。\$\$ 代表产生式左部的值，\$n 为产生式右部第 n 个 token 的值，**注意运算符等字符也计算其中**，yygettoken 函数部分作用是实现词法分析功能，也就是本次作业着重修改的部分。89-98 行为 main 函数，需要注意 92 行需要与 parser name 相对应。

3.4 Yacc 程序修改——简单表达式计算

首先进行第一条作业要求的修改，以加号为例，用定义名称为 ADD 的 token 表示加号，首先需要声明该 token 然后在使用加号的位置修改为 ADD，最后在词法分析函数 yygettoken 写明加号的情况如何处理。

```

1  ...
2  // place any declarations here
3  %token ADD
4  %left ADD '-'
5  %left '*' '/'
6  %right UMINUS
7  ...
8  expr :      expr ADD expr    { $$ = $1 + $3; }
9  ...
10 int YYPARSENAME::yygettoken()
11 {

```

```
12 // place your token retrieving code here
13 int t;
14 t=getchar();
15 if(t=='+')
16     return ADD;
17 else
18     return t;
19 //return getchar();
20 }
21 ...
```

其余运算符及整数请同学们举一反三实现。

之后修改实现的是识别忽略空白字符功能以及可识别多位十进制数字，思路是参考 PPT 的 95-96 页程序，对空格、制表符、回车和数字分情况处理。需要注意的是以下几点：

- a 回车在讲义原 yacc 程序中语法分析中 lines 的产生式中，我们需要处理这个部分，由于要实现忽略回车，因此我们可使用分号去替换 lines 产生式中的 `'\n'`，这样在测试程序时，简单表达式以分号作结而不是以回车作结，就可以起到忽略回车的作用。
- b 有关于 PPT 中 NUM 的宏定义问题，在 yacc 中由于声明的 token 都会有一个互不冲突的整数值，因此不需要在 yacc 中进行 NUMBER 的宏定义，声明 NUMBER 的 token 即可。
- c PPT 中 lineno 用于记录行数，对于完成作业功能来说并无意义，可省略该变量将回车符与制表符空格统一起来。tokenval 记录多位十进制整数的值，由于需要在语法分析中使用，所以需要声明在 class definition 部分，由于 YYSTYPE 为 double 类型，因此将 tokenval 声明为 double 类型。将原 yacc 代码 NUMBER 对应一位整数的上下文无关文法及翻译模式注释掉，将 NUMBER 视为终结符，添加执行动作，将 tokenval 的值赋给 NUMBER。
- d ungetc 函数作用是将读出的数据再次放回到缓冲区去，下一次读数据时，会再次读出来的。由于判断处理多位十进制数字时，需要读到最新一位为非数字时停止，但这个非数字字符已被读出，因此需要再次放回缓冲区，所以同学们写代码时不要忽略。
- e 函数对应的头文件不要忘记添加。

完成第二条作业要求后的代码修改如下：

```

1  ...
2  // class definition
3  {
4      // place any extra class members here
5      virtual int yygettoken();
6      double tokenval;
7  }
8  ...
9  // place any declarations here
10 %token NUMBER
11 %token ADD
12 %left ADD '-'
13 %left '*' '/'
14 %right UMINUS
15 ...
16 lines :      lines expr ';' { cout << $2 << endl; }
17       |
18       |
19       ;
20
21 expr  :      expr ADD expr  { $$ = $1 + $3; }
22       ...
23       |      NUMBER        { $$ = tokenval; }
24       ;
25
26 /*
27 NUMBER :      '0'                { $$ = 0.0; }
28         |      '1'                { $$ = 1.0; }
29         |      '2'                { $$ = 2.0; }
30         |      '3'                { $$ = 3.0; }
31         |      '4'                { $$ = 4.0; }
32         |      '5'                { $$ = 5.0; }
33         |      '6'                { $$ = 6.0; }
34         |      '7'                { $$ = 7.0; }
35         |      '8'                { $$ = 8.0; }
36         |      '9'                { $$ = 9.0; }
37         ;
38 */
39 %%
40 ...
41 int YYPARSENAME::yygettoken()
42 {
43     // place your token retrieving code here
44     int t;
45     while (1) {
46         t = getchar();
47         if (t == ' ' || t == '\t' || t == '\n')
48             ;
49         else if (isdigit(t)) {
50             tokenval = 0;
51             while (isdigit(t)) {
52                 tokenval = tokenval * 10 + t - '0';
53                 t = getchar();
54             }
55             ungetc(t, stdin);
56             return NUMBER;
57         }
58         else if (t == '+')
59             return ADD;
60         else { tokenval = NULL; return t; }
61     }
62     //return getchar();
63 }

```

64 | ...

前两条作业要求修改后效果如下 (测试表达式为 $2+672/4$ 和 $32-3*5$):



```
C:\Users\py199\Desktop\TestPG\Release\TestPG.exe
2 +672
/4
;
170
32 -
3 *
5;
17
```

3.5 中缀表达式转后缀表达式

由于需要返回的是一个后缀表达式, 因此 `$$` 变量类型需要修改为字符串类型, `YYSTYPE` 声明为 `char *`, 而词法分析函数中不仅需要分析运算符, 多位十进制整数, 空白字符, 还需要识别标识符 `ID`。而对于多位十进制整数, 不再需要得到整数值, 而是需要得到整数对应的字符串。对于多位十进制整数, 思路是对应 `tokenval` 声明一个全局的字符串变量, 比如 `py`, 用于赋给 `NUMBER` 值, 当读到一个字符为整数字符时, 连续读接下来的数字字符直至读到不为数字字符的字符, 调用 `ungetc` 函数将读出的非数字字符放回缓冲区, 将读到的若干数字字符存为一个字符串, 将这个字符串的值赋给 `py`。`NUMBER` 的翻译模式执行动作就修改为将 `py` 的值赋给 `NUMBER.ID` 与 `NUMBER` 思路类似, 声明一个名为 `ID` 的 `token` 和一个全局的字符串变量, 比如 `lxy`, 用于赋给 `ID` 值, 由于标识符是有数字、字母和下划线组成的, 而第一个字符不可以为数字, 因此思路为当读到一个字符为字母或下划线时, 连续读接下来的字符, 直到出现一个不是数字或字母或下划线的字符停止, 调用 `ungetc` 函数将读出的非数字非字母非下划线字符放回缓冲区, 将读到的若干字符存为一个字符串, 将这个字符串的值赋给 `lxy`。`ID` 的翻译模式执行动作就修改为将 `lxy` 的值赋给 `ID`。对于加减乘除括号负号空白字符的词法分析部分不需修改, 但语法分析部分需要将计算值修改成字符串连接。同样的, 接下来给出的代码只给出加法部分, 其余请同学们补充。

```
1 ...
2 // declarations section
3 %include {
4 #ifndef YYSTYPE
5 #define YYSTYPE char *
```

```

6  #endif
7  }
8  ...
9  // class definition
10 {
11     // place any extra class members here
12     virtual int yygettoken();
13     string py;
14     string lxy;
15 }
16 ...
17 // place any declarations here
18 %token NUMBER
19 %token ID
20 %token ADD
21 %left ADD '-'
22 %left '*' '/'
23 %right UMINUS
24 ...
25 expr :      expr ADD expr  { $$ = new char[50]; strcpy($$, $1); strcat($$, $3); strcat($$, "+"); }
26     ...
27     |      NUMBER          { $$ = new char[50]; strcpy($$, py.c_str()); }
28     |      ID              { $$ = new char[50]; strcpy($$, lxy.c_str()); }
29     ;
30 ...
31 int YYPARSENAME::yygettoken()
32 {
33     // place your token retrieving code here
34     int t;
35     while (1) {
36         t = getchar();
37         if (t == ' ' || t == '\t' || t == '\n')
38             ;
39         else if (isdigit(t)) {
40             string temp="";
41             while (isdigit(t)) {
42                 temp.append(1, t);
43                 t = getchar();
44             }
45             py = temp;
46             ungetc(t, stdin);
47             return NUMBER;
48         }
49         else if (( ( t >= 'a' && t <= 'z' ) || ( t >= 'A' && t <= 'Z' ) || ( t == '_' ) )){
50             string temp2="";
51             while (( ( t >= 'a' && t <= 'z' ) || ( t >= 'A' && t <= 'Z' )
52                 || ( t == '_' ) || isdigit(t) ) ) {
53                 temp2.append(1, t);
54                 t = getchar();
55             }
56             lxy = temp2;
57             ungetc(t, stdin);
58             return ID;
59         }
60         else if (t == '+')
61             return ADD;
62         else { return t; }
63     }
64 }
65 ...

```

第三条作业要求修改后效果如下 (测试表达式为 $2+A_a2b*(1234_haha)$):



```
C:\Users\py199\Desktop\TestPG1\Release\TestPG.exe
2 + A_a2b
*
( 1234 -
_haha)
;
2A_a2b1234_haha-*+
```

3.6 思考——符号表

本部分鼓励学有余力的同学们进行探究实践，不做硬性要求。

3.6.1 思考题要求

在第二条作业要求的基础上，实现功能更强的词法分析和语法分析程序，使之能支持变量，修改词法分析程序，能识别变量（标识符）和“=”符号，修改语法分析器，使之能分析、翻译“a=2”形式的（或更复杂的，“a=表达式”）赋值语句，当变量出现在表达式中时，能正确获取其值进行计算（未赋值的变量取 0）。当然，这些都需要实现符号表功能。

3.6.2 思考题提示

可以使用数组，vector，map 等数据结构实现符号表。词法分析部分符号表思路可参考 PPT 的 98-99 页，标识符的实现已在第三条作业要求实现过程中讲解，“=”的识别可类比别的运算符，语法分析部分需要添加相应赋值语句，修改上下文无关文法及翻译模式。