bI dymanic language system

© Dmitry Ponyatov <dponyatov@gmail.com>

January 4, 2016

Contents

Intro

Any program must have scripting ability for configs and user extensions. bI system provides universal script engine for bI language dialect and dynamic data types C^{++} class tree for internal use in generated program. I was impressed by $SmallTalk$ system ideology, bI system follows this way to gui-powered interactive system for translators design, symbolic computations and CAD/CAM/EDA environment.

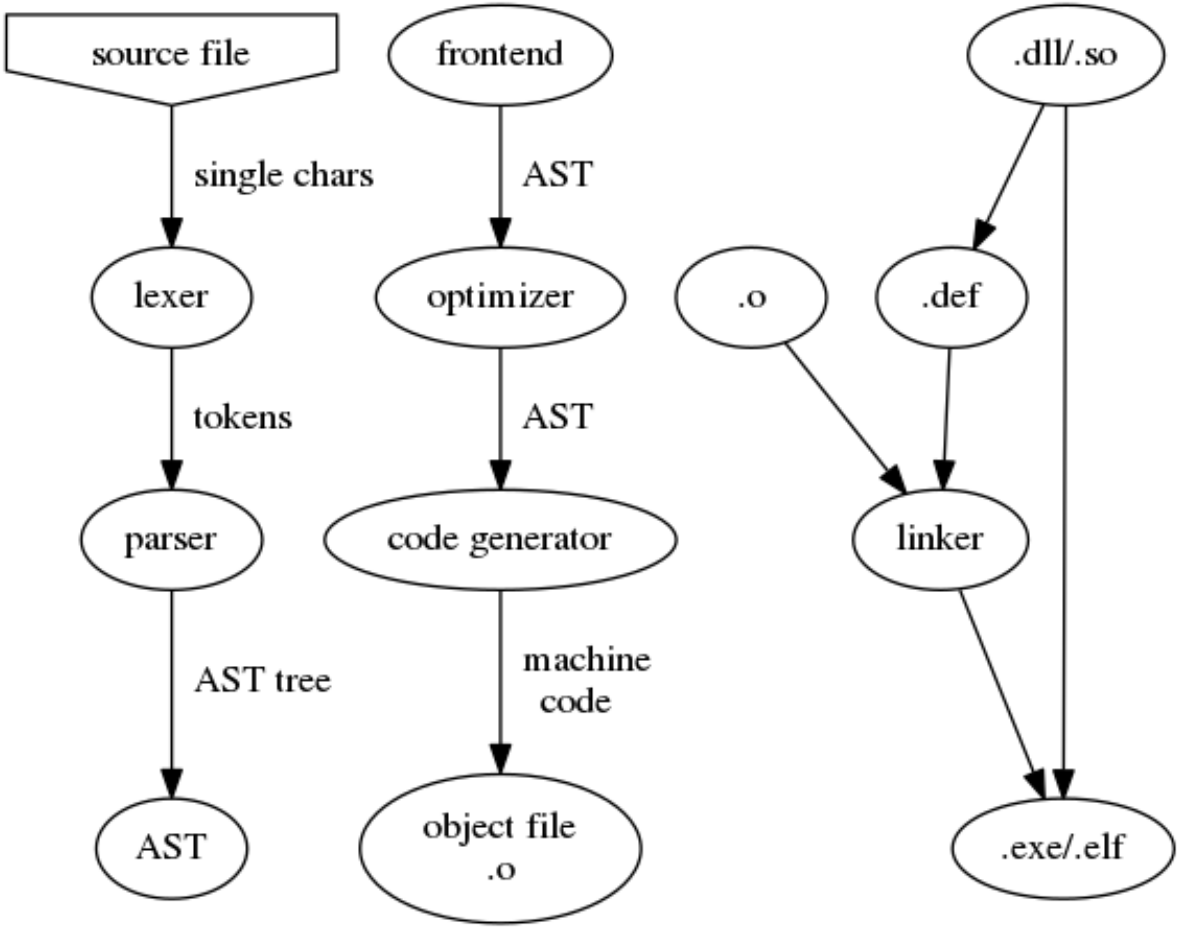
Goals

- metaprogramming, computer language design and translator development
- symbolic and numeric computations
- clustering and cloud computing
- complex engineering systems design
- statical translation to $C^{++}/Java$ for multiplatform software development ($\boxplus Windows/Linux/Android$)

Files

ypp.ypp	flex	parser 1.2
lpp.lpp	bison	lexer 1.1
hpp.hpp	C^{++}	headers 1.3
cpp.cpp	C^{++}	core **??**
Makefile	make	build script
rc.rc	windres	win32 resource description
bat.bat		win32 (g)gvim helper
doc/	LaTeX	manual
doc/Makefile		
doc/bI.pdf		

Compiler structure



Chapter 1

Core system

1.1 Lexer /lpp.lpp/

Lexer uses **flex** generator, produces **lex.yy.c**.

All defines moved to **hpp.hpp**, lexer header includes buffer for string parsing.

lpp.lpp

```
1 %{
2 #include "hpp.hpp"
3 std::string StringLexBuffer;
4 %}
```

Options disables `yywrap()` function usage and enables line number autocount for error reporting.

lpp.lpp

```
%option noyywrap
%option yylineno
```

Rules section described part by part in scalar types 1.5 manual sections.

lpp.lpp

```
%%
...
```

Unused chars will be dropped by this rules at end of lexer:

lpp.lpp

```
[ \t\r\n]+      {}      /* spaces */
.               {}      /* undetected chars */
%%
```

Lexer C^{++} API includes this objects: `TOC()` macro used in lexer rules, creates

hpp.hpp

```
                                                    // == lexer interface ==
extern int yylex();                                 // parse next token
extern int yylineno;                                // current source line
extern char* yytext;                                // found token text
#define TOC(C,X) { yylval.o = new C(yytext); return X; }
```

hpp.hpp

```
                                                    // == parser interface ==
extern int yyparse();                               // run parser
extern void yyerror(std::string);                   // error callback
```

1.2 Parser /ypp.ypp/

Core parser uses **bison** for **ypp.tab.cpp**, **ypp.tab.hpp**

Parser header looks like lexer header, all defines done in **hpp.hpp**.

ypp.ypp

```
1 %{
2 #include "hpp.hpp"
3 %}
```

1.3 Header file /hpp.hpp/

Header file contents wrapped by include-once preprocessor hint:

hpp.hpp

```
1 #ifndef _H_bl
2 #define _H_bl
3 #endif // _H_bl
```

Standard C^{++} includes used in core:

hpp.hpp

```
1                                                    // == std.includes ==
2 #include <iostream>
3 #include <sstream>
4 #include <cstdlib>
```

```
5 #include <vector>
6 #include <map>
```

1.4 Comments

1.4.1 Line comment

lpp.lpp

```
1 #[^\n]*          {}        /* line comment */
```

1.5 Scalars