哈爾濱工業大學（威海）

Harbin Institute of Technology at Weihai

编译原理实验报告

院　　系：计算机科学与技术学院

班　　级：

姓　　名：李彦哲

学　　号：

指导教师：闫健恩

哈尔滨工业大学（威海）

实验报告撰写规范

1） 正文使用五号字，单倍行距，中文使用宋体，英文使用 **times new roma** 字体；
2） 表格使用三线表，必须有表格标题；
3） 所有示意图必须使用 **visio、word** 等工具生成，不得手工画图或从其他资料中截图，并且每个图要有标题和编号。图中文字不得大于正文文字；
4） 注意行文的规范，段落空行等；
5） 报告采用双面打印，将 **3** 个实验一起装订，使用左侧装订方式；
6） 实验报告电子版请提供 **pdf** 格式。

《编译原理》实验成绩评定表

实验一　词法扫描器设计（共 20 分）			
评审项目	评审内容	成绩标准	成绩
实验内容 完成情况	实验内容是否全部完成； 程序功能是否完整； 实验结果是否准确； 调试分析是否完整等	10 分	
系统设计	系统整体功能的分析设计与说明	5 分	
报告内容完整性	文档格式规范性： 文字是否规范； 图表是否规范； 术语是否准确等	5 分	
实验二　LR 语法分析器设计（共 40 分）			
评审项目	评 审 内 容	成绩标准	成绩
实验内容 完成情况	实验内容是否全部完成； 程序功能是否完整； 实验结果是否准确； 调试分析是否完整等	20 分	
系统设计	系统整体功能的分析设计与说明	10 分	
报告内容 完整性	文档格式规范性： 文字是否规范； 图表是否规范； 术语是否准确等	10 分	
实验三　语义分析及中间代码生成（共 40 分）			
评审项目	评 审 内 容	成绩标准	成绩
实验内容 完成情况	实验内容是否全部完成； 程序功能是否完整； 实验结果是否准确； 调试分析是否完整等	20 分	
系统设计	系统整体功能的分析设计与说明	10 分	
报告内容 完整性	文档格式规范性： 文字是否规范； 图表是否规范； 术语是否准确等	10 分	
总体评价		总成绩	

实验一 词法扫描器设计

一 实验目的

通过设计调试词法分析程序，实现从源程序中分出各种单词的方法；加深对课堂教学的理解；提高词法分析方法的实践能力。

二 实验内容

设计一个简单的类 C 语言的词法扫描器。

三 实验要求

（一） 程序设计要求

（1） 根据附录给定的文法，从输入的类 C 语言源程序中，识别出各个具有独立意义的单词，即关键字、标识符、常数、运算符、分隔符五大类；文法见最后附录。

（2） 提供源程序输入界面；

（3） 词法分析后可查看符号表和 TOKEN 串表；

（4） 保存符号表和 TOKEN 串表（如：文本文件）；

（5） 遇到错误时可显示提示信息，然后跳过错误部分继续进行分析。

（二）实验报告撰写要求

（1） 系统功能分析设计（包括各个子功能模块的功能说明）；

（2） 设计方案；

1） 功能模块结构图；

2） 主程序流程图；

3） 主要子程序的流程图（若有必要）；

4） 主要数据结构：符号表、TOKEN 串表等。

5） 开发平台（操作系统、设计语言）；

（3） 具体实现（包括主控程序、各个功能模块的具体实现，给出主要函数部分即可，不用粘贴全部代码）。

（4） 实验总结

（1） 系统功能分析设计

本实验主要分为两个模块。

第一个模块是公共的单词识别器抽象接口。每种单词实现一个对应的的实现类。封装后的识别器被递交给统一的主控进行管理。

第二个模块是主控模块。统一接管输入输出流，识别窗口管理等内容。各识别器需要事先被注册到主控模块中，其注册顺序决定了匹配的优先级。

主控内部维护一个识别缓冲区，用于存储当前正在进行被识别的字符串，称之为有效识别窗口。该窗口定义为：两段空白符中间的非空白符片段。

字符串缓冲区设有两个尾部指针，分别表示当前已经匹配出的子串的最大长度和当前窗口尾部。

匹配逻辑可简要总结为以下几点：

每一次循环，都读入一个有效字符，顺序调用所有识别器。

只要其中之一返回有效值，且匹配的串长度大于当前最大匹配长度，则将当前候选单词类型更新为此识别器的结果，并不再调用后续识别器。直接进入下一个循环。

循环的退出条件是当前窗口已经读取结束。主控将返回最后的候选单词类型。并清空缓冲区中已经匹配出来的串。

于是此时缓冲区可能为两种状态：

A. 之前窗口仍剩下剩余未匹配完的串

B. 之前窗口全部匹配完毕

当下一次调用 **YYLex()**时：

如果处于 **A** 状态，主控会继续识别剩余的串，而不会读取新的字符。直到当前窗口全部匹配完。如果多次词法分析过程仍然无法匹配剩余的串，说明词法定义有误，无法识别。

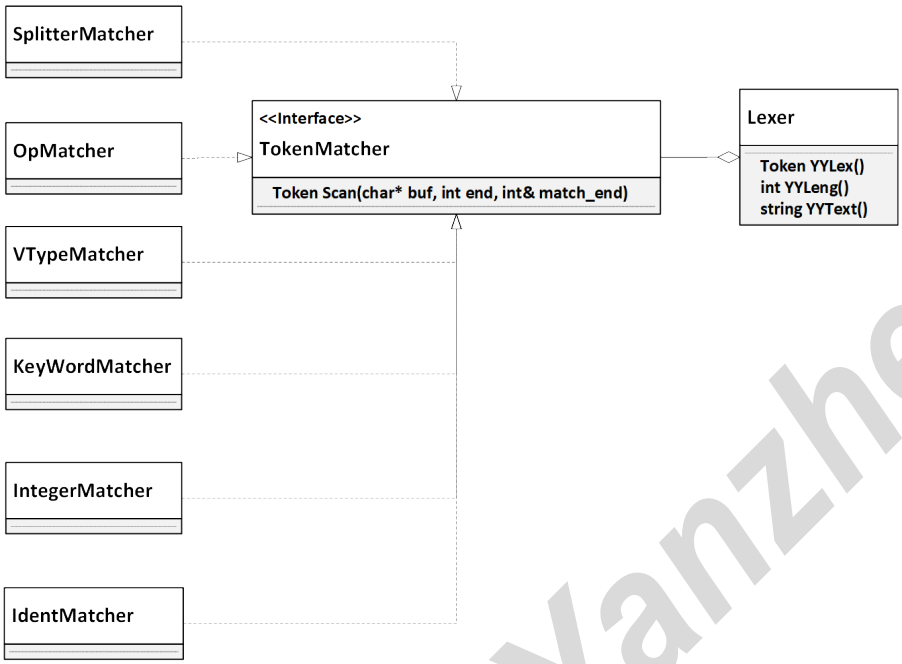
如果处于 **B** 状态，则继续重复 **1)**过程。

本实验完整代码可以在我的 **Github** 主页查看

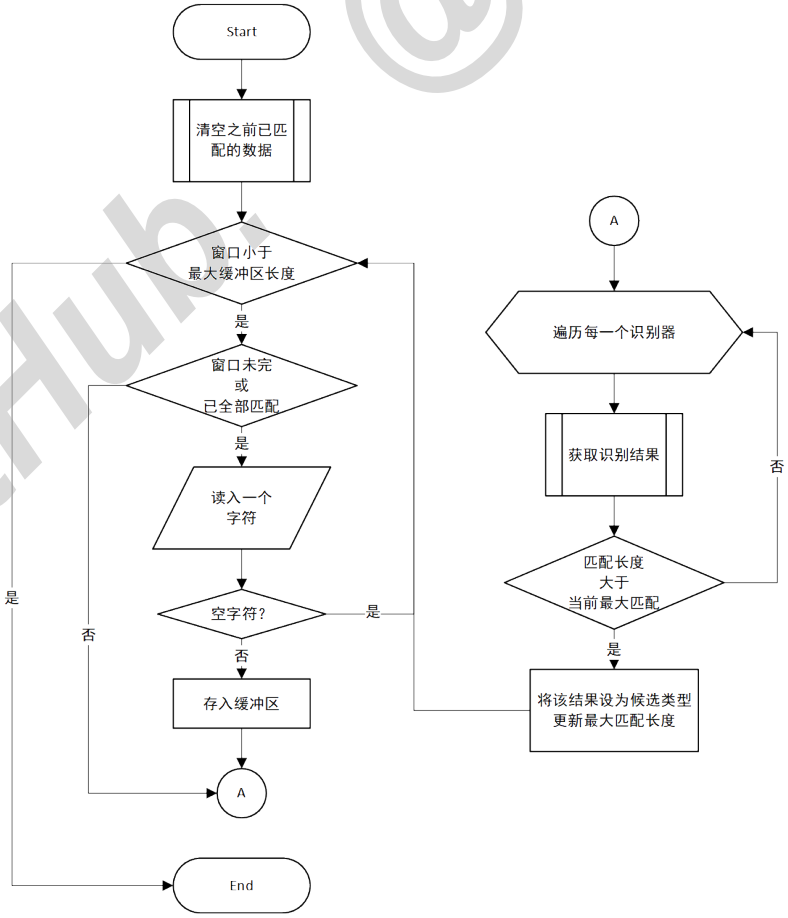
https://github.com/YanzheL/hit-compiler-lab

（2） 设计方案

1) 功能模块结构图



2) 主程序流程图



3) 主要数据结构

符号表

```
struct SymInfo {
  std::string name;
  ValTypeEnum type;
  void *value;
  std::string ToString() const;
};

// 符号表接口，原意是想用多个子类对符号表结构进行多种实现，比如用搜索树实现
class SymTable {
 public:
  virtual int insert(std::string name, ValTypeEnum type, void *val) = 0;
  virtual SymInfo *lookup(int id) = 0;
};

// 暂时只用了最简单的方式
class SimpleSymTable : public SymTable {
 public:
  int insert(std::string name, ValTypeEnum type, void *val);
  SymInfo *lookup(int id);
 private:
  std::vector<SymInfo> sym_table_;
};
```

TOKEN 类型

```
enum TokenType {
  TK_KEYWORD,
  TK_ID,
  TK_CONST,
  TK_OP,
  TK_SPLITTER,
  TK_VTYPE,
  TK_UNKNOWN
};
```

4) 开发平台

操作系统：Ubuntu 18.04

开发语言：C++ 11

依赖库：无

（3） 具体实现

本实验所有代码都为本人原创，独立完成。

主控类

```
class Lexer {
 public:
  const static int BUFFER_LEN = 100;
  Lexer() : in_{&std::cin}, match_end_{0}, bf_end_{0} {}
  Lexer(std::istream *in) : in_{in}, match_end_{0}, bf_end_{0} {}
  // 把当前缓冲区中的字符穿作为 string 返回
  inline std::string YYText() const { return {buffer_, buffer_ + match_end_}; }
  // 返回当前缓冲区中的串长度
  inline int YYLeng() const { return match_end_; }
  // 注册识别器到此分析器中
  inline void RegisterMatcher(TokenMatcher *scanner) { scanners_.push_back(scanner); }
  // 主入口
  TokenType YYLex() {
    _CleanBuffer();
    TokenType result = TK_UNKNOWN;
    while (bf_end_ <= BUFFER_LEN) {
      // 如果窗口还没有读完整，或者窗口读完了，但缓冲区也空了(说明之前的串全部被识别掉了)，则读入下一个字符
      if (!tk_finished_ || (tk_finished_ && bf_end_ == 0)) {
        tk_finished_ = false;
        char c = in_->get();
        if (_IsDelim(c)) {
          // 发现是个空白符
          if (bf_end_ != 0) {
            // 如果缓冲区非空，说明这是窗口尾部的空白符，表示这个窗口读完整了
            tk_finished_ = true; break;
          } else // 否则说明还没有读到有效字符，则跳过
            continue;
        }
        // 此时的 c 必然是有效字符
        buffer_[bf_end_++] = c;
      } else if (match_end_ != 0) { // 此时说明窗口是完整的，识别出来的串也结束了，那么就返回给调用方吧
        break;
      }
      // 对当前缓冲区中的串按照识别器优先级进行扫描
      for (TokenMatcher *scanner:scanners_) {
        int cur_mend = 0; // 用来存储当前识别器对当前缓冲区中的串的最大匹配长度
        TokenType type = scanner->Scan(buffer_, bf_end_, cur_mend);
        if (cur_mend > match_end_) {
          // 如果当前的匹配长度比以前的都长，说明这次匹配的更合适
```

```
40.           // 则把候选类型设置为当前的结果
41.           result = type; match_end_ = cur_mend;
42.           // 一旦当前的结果成为新的候选，则不再调用后续识别器，否则高优先级结果会被低优先级结果覆盖
43.           break;
44.         }
45.       }
46.     }
47.     return result; // 默认返回识别失败
48.   }
49.  private:
50.   // 输入流
51.   std::istream *in_;
52.   // 当前缓冲区的末尾
53.   int bf_end_;
54.   // 当前缓冲区中已经被匹配出来的子串的末尾(长度)
55.   int match_end_;
56.   /**
57.    * 我们把从空白符开始，开始读有效字符，然后再读到空白符视为一个完整的识别窗口
58.    * @param tk_finished_ 表示该窗口是否读完
59.    */
60.   bool tk_finished_;
61.   char buffer_[BUFFER_LEN]; // 识别缓冲区，存放当前窗口正在被识别的串
62.   // 空白符
63.   const char *delims_ = "\n\t \r";
64.   // 被注册到此词法分析器中的所有识别器，按照优先级排列
65.   std::vector<TokenMatcher *> scanners_;
66.   // 把缓冲区已经识别出来的串清除，再把后面剩余的串移动到最前面
67.   inline void _CleanBuffer() {
68.     int n = bf_end_ - match_end_;
69.     std::memcpy(buffer_, buffer_ + match_end_, n);
70.     bf_end_ = n; match_end_ = 0;
71.   }
72.   // 判断 c 是否是空白符
73.   inline bool _IsDelim(char c) const {
74.     for (int i = 0; i < strlen(delims_); ++i)
75.       if (delims_[i] == c) return true;
76.     return false;
77.   }
78. };
```

常整数识别器

```
class IntegerMatcher : public TokenMatcher {
 public:
  TokenType Scan(const char *buf, int end, int &match_end) override {
    if (end == 0) return TK_UNKNOWN;
    int i = 0;
    int state = 0;
    TokenType result = TK_CONST;
    while (i < end) {
      switch (state) {
        case 0:
          if (IsDigit(buf[i])) {
            state = 1;
            ++i;
            match_end = i;
          } else
            state = 2;
          break;
        case 1:
          if (IsDigit(buf[i])) {
            state = 1;
            ++i;
            match_end = i;
          } else
            state = 2;
          break;
        case 2:
          if (match_end == 0) {
            result = TK_UNKNOWN;
          }
          goto finish;
      }
    }
    finish:
    return result;
  }
 private:
  inline static bool IsDigit(char c) { return c >= '0' && c <= '9'; }
};
```

操作符识别器

```
class OpMatcher : public TokenMatcher {
 public:
  TokenType Scan(const char *buf, int end, int &match_end) override {
    if (end == 0) return TK_UNKNOWN;
    int i = 0; int state = 0;
    TokenType result = TK_OP;
    while (i < end) {
      switch (state) {
        case 0:
          if (InArray(buf[i], type1_)) {
            state = 1;
            ++i;
            match_end = i;
          } else if (InArray(buf[i], type2_)) {
            state = 2;
            ++i;
            match_end = i;
          } else
            state = 3;
          break;
        case 1:state = 3;
          break;
        case 2:
          if (buf[i] == '=') {
            state = 1;
            ++i;
            match_end = i;
          } else
            state = 3;
          break;
        case 3:
          if (match_end == 0)
            result = TK_UNKNOWN;
          goto finish;
      }
    }
    finish:
    return result;
  }
 private:
  const char *type1_ = "+-*/";
  const char *type2_ = "<>=";
};
```

分隔符识别器

```
class SplitterMatcher : public TokenMatcher {
 public:
  TokenType Scan(const char *buf, int end, int &match_end) override {
    if (end == 0) return TK_UNKNOWN;
    TokenType result = TK_SPLITTER;
    if (!InArray(buf[0], splitters_)) result = TK_UNKNOWN;
    else match_end = 1;
    return result;
  }
 private:
  const char *splitters_ = "();'";
};
```

关键字识别器

```
class KeyWordMatcher : public TokenMatcher {
 public:
  KeyWordMatcher() : keywords_{"if", "then", "else", "while", "do"}, successType_{TK_KEYWORD} {}
  KeyWordMatcher(std::vector<std::string> keywords, TokenType successType) : keywords_{std::move(keywords)},
                                                                              successType_{successType} {}
  TokenType Scan(const char *buf, int end, int &match_end) override {
    if (end == 0) return TK_UNKNOWN;
    std::string str(buf, buf + end);
    for (const auto &kw:keywords_) {
      if (str == kw) {
        match_end = kw.length();
        return successType_;
      }
    }
    return TK_UNKNOWN;
  }
 private:
  const std::vector<std::string> keywords_;
  TokenType successType_;
};
```

标识符识别器

```
class IdentMatcher : public TokenMatcher {
 public:
  TokenType Scan(const char *buf, int end, int &match_end) override {
    if (end == 0) return TK_UNKNOWN;
    int i = 0;
    int state = 0;
    TokenType result = TK_ID;
    while (i < end) {
      switch (state) {
        case 0:
          if (InArray(buf[i], type1_)) {
            state = 1;
            ++i;
            match_end = i;
          } else
            state = 2;
          break;
        case 1:
          if (InArray(buf[i], type2_)) {
            state = 1;
            ++i;
            match_end = i;
          } else
            state = 2;
          break;
        case 2:
          if (match_end == 0) result = TK_UNKNOWN;
          goto finish;
      }
    }
    finish:
    return result;
  }
 private:
  const char *type1_ = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
  const char *type2_ = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
};
```

类型符识别器

```
class VTypeMatcher : public KeyWordMatcher {
 public:
  VTypeMatcher() : KeyWordMatcher({"int", "long", "double", "float"}, TK_VTYPE) {}
};
```

识别器接口类

```
class TokenMatcher {
 public:
  virtual TokenType Scan(const char *buf, int end, int &match_end) = 0;
 protected:
  inline static bool InArray(char c, const char *arr) {
    for (int i = 0; i < strlen(arr); ++i)
      if (arr[i] == c)
        return true;
    return false;
  }
};
```

（4） 实验结果

测试用例

```
int a;
int b;
int c;
a=2;
b=1;
if (a>b)
  c=a+b;
else
  c=a-b;
```

程序输出

```
----------词法分析结果----------
(  类型符  ,    int  )
(  标识符  ,    a  )
(  分隔符  ,    ;  )
(  类型符  ,    int  )
(  标识符  ,    b  )
(  分隔符  ,    ;  )
(  类型符  ,    int  )
(  标识符  ,    c  )
(  分隔符  ,    ;  )
(  标识符  ,    a  )
(  运算符  ,    =  )
(  常整数  ,    2  )
(  分隔符  ,    ;  )
(  标识符  ,    b  )
(  运算符  ,    =  )
(  常整数  ,    1  )
(  分隔符  ,    ;  )
(  关键字  ,    if  )
(  分隔符  ,    (  )
(  标识符  ,    a  )
(  运算符  ,    >  )
(  标识符  ,    b  )
(  分隔符  ,    )  )
(  标识符  ,    c  )
(  运算符  ,    =  )
(  标识符  ,    a  )
(  运算符  ,    +  )
(  标识符  ,    b  )
(  分隔符  ,    ;  )
(  关键字  ,    else  )
(  标识符  ,    c  )
(  运算符  ,    =  )
(  标识符  ,    a  )
(  运算符  ,    -  )
(  标识符  ,    b  )
(  分隔符  ,    ;  )
----------符号表----------
<  TYPE  ,  NAME  ,  VAL  >
(  UNKNOWN  ,    a  ,  NULL  )
(  UNKNOWN  ,    b  ,  NULL  )
(  UNKNOWN  ,    c  ,  NULL  )
```

测试用例

```
int m;
int n;
double c;
while (m>n)
  if (c>0)
    while (m<c)
      n = m - 1;
  else
    n = m - 2;
```

程序输出

```
---------- 词法分析结果 ----------
(  类型符  ,    int  )
(  标识符  ,    m  )
(  分隔符  ,    ;  )
(  类型符  ,    int  )
(  标识符  ,    n  )
(  分隔符  ,    ;  )
(  类型符  ,  double  )
(  标识符  ,    c  )
(  分隔符  ,    ;  )
(  关键字  ,  while  )
(  分隔符  ,    (  )
(  标识符  ,    m  )
(  运算符  ,    >  )
(  标识符  ,    n  )
(  分隔符  ,    )  )
(  关键字  ,   if  )
(  分隔符  ,    (  )
(  标识符  ,    c  )
(  运算符  ,    >  )
(  常整数  ,    0  )
(  分隔符  ,    )  )
(  关键字  ,  while  )
(  分隔符  ,    (  )
(  标识符  ,    m  )
(  运算符  ,    <  )
(  标识符  ,    c  )
(  分隔符  ,    )  )
(  标识符  ,    n  )
(  运算符  ,    =  )
(  标识符  ,    m  )
(  运算符  ,    -  )
(  常整数  ,    1  )
(  分隔符  ,    ;  )
(  关键字  ,  else  )
(  标识符  ,    n  )
(  运算符  ,    =  )
(  标识符  ,    m  )
(  运算符  ,    -  )
(  常整数  ,    2  )
(  分隔符  ,    ;  )
---------- 符号表 ----------
<  TYPE  ,  NAME  ,  VAL  >
(  UNKNOWN  ,    m  ,  NULL  )
(  UNKNOWN  ,    n  ,  NULL  )
(  UNKNOWN  ,    c  ,  NULL  )
```

实验二 LR语法分析器设计

一 实验目的

通过设计调试 LR 语法分析程序，实现根据词法分析的 TOKEN 字，进行文法的语法分析；加深对课堂教学的理解；提高语法分析方法的实践能力。

二 实验内容

使用附录中的文法，可以对类似下面的程序语句进行语法分析：

```
int a;
int b;
int c;
a=2;
b=1;
if （a>b）
    c=a+b;
 else
    c=a-b;
```

三 实验要求

（一） 程序设计要求

（1）给出主要数据结构：分析栈、状态等；

（2）将词法扫描器作为一个子程序，每次调用返回一个 TOKEN；

（3）程序界面：基本要求包含源程序输入功能、语法分析的结果表示（文件或者图形方式）；其他过程表示形式可以自行设计。

（二）实验报告撰写要求

（1） 系统功能分析设计（包括各个子功能模块的功能说明）；

（2） 设计方案；

1） 功能模块结构图；

2） 主程序流程图；

3） 主要子程序的流程图（若有必要）；

4） 主要数据结构：分析栈、状态、分析表等。

5） 开发平台（操作系统、设计语言）；

（3） 具体实现（包括主控程序、各个功能模块的具体实现，给出主要函数部分即可，不用粘贴全部代码）。

（4） 实验总结（程序调试结果以及实验心得体会等）

（1） 系统功能分析设计

本实验使用 FLEX 结合 Bison 实现。Flex 是 Lex 的更新版本，Bison 是 Yacc 的更新版本。新旧版本语法相似。

本实验主要实现流程为：

1. 编写 lex 文件，进而生成词法分析器 lexer.cc
2. 编写 Bison 文件 parser.y，可以得到语法分析器文件 parser.h 和 parser.cc 文件。

因为本实验不涉及语义分析，因此每个 AST 节点都使用同样的结构，是一个多分支节点。节点的构造随着 BISON 语法分析的进行而逐步完成。

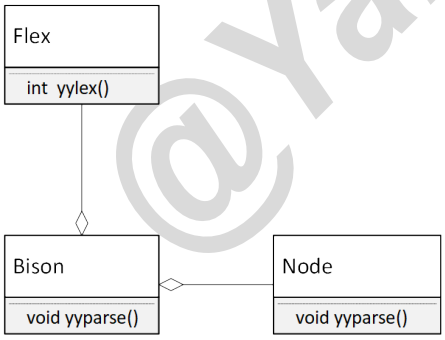
在构造完整棵 AST 树后，为了方便展示，此 AST 树被序列化成 Dot 图文件格式，再使用 Graphviz 工具将其以图片形式展现出来。

本实验完整代码可以在我的 Github 主页查看

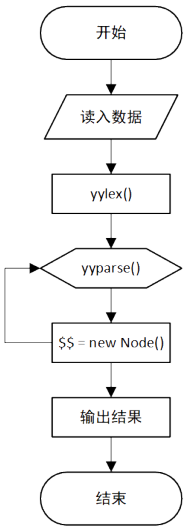
https://github.com/YanzheL/hit-compiler-lab

（2） 设计方案

1) 功能模块结构图



2) 主程序流程图



3) 主要数据结构

AST 节点

```
class Node {
 public:
  int idx = -1;
  std::string text;
  int token;
  std::vector<Node *> children;
 public:
  explicit Node(int token) : token(token) {}
  // 辅助函数，把当前的 AST 树序列化成 dot 类型的文件字符串，后续用图形化工具显示
  std::string ToDot() const;
  // 辅助函数，取得当前 AST 的所有边
  void dump_edges(const Node *parent, std::vector<std::pair<const Node *, const Node *>> &
res) const;
  std::string name() const;
};
```

4) 开发平台

操作系统：Ubuntu 18.04

开发语言：C++ 11

依赖库：Flex, Bison

（3） 具体实现

本实验所有代码都为本人原创，独立完成。

Flex 文件 lexer.l

```
%option noyywrap
%{
#include <iostream>
#include <string>
#include "node.h"
#include "parser.h"
#define ADD_LEAF(t) {              \
  auto leaf = new Node(t);         \
  leaf->text = yylval->text;       \
  yylval->children.push_back(leaf);\
}
#define TOKEN(t) {                           \
  yylval = new Node(t);                      \
  yylval->text = std::string(yytext, yyleng); \
}
%}

%%

[ \t\n]                 ;
int|float               TOKEN(TTYPE) ADD_LEAF(TTYPE) return TTYPE;
if                      TOKEN(TIF) return TIF;
then                    TOKEN(TTHEN) return TTHEN;
else                    TOKEN(TELSE) return TELSE;
while                   TOKEN(TWHILE) return TWHILE;
do                      TOKEN(TDO) return TDO;
[a-zA-Z_][a-zA-Z0-9_]*  TOKEN(TIDENTIFIER) ADD_LEAF(TIDENTIFIER) return TIDENTIFIER;
[0-9]+                  TOKEN(TINTEGER) ADD_LEAF(TINTEGER) return TINTEGER;
"="                     TOKEN(TEQUAL) return TEQUAL;
"=="                    TOKEN(TCEQ) return TCEQ;
"!="                    TOKEN(TCNE) return TCNE;
"<"                     TOKEN(TCLT) return TCLT;
"<="                    TOKEN(TCLE) return TCLE;
">"                     TOKEN(TCGT) return TCGT;
">="                    TOKEN(TCGE) return TCGE;
"("                     TOKEN(TLPAREN) return TLPAREN;
")"                     TOKEN(TRPAREN) return TRPAREN;
"."                     TOKEN(TDOT) return TDOT;
"'"                     TOKEN(TSQUOTE) return TSQUOTE;
"+"                     TOKEN(TPLUS) return TPLUS;
"-"                     TOKEN(TMINUS) return TMINUS;
```

```
42. "*"                     TOKEN(TMUL) return TMUL;
43. "/"                     TOKEN(TDIV) return TDIV;
44. ";"                     TOKEN(TSEMI) return TSEMI;
45. .                       std::cout<< "Unknown token!" << std::endl; yyterminate();
46.
47. %%
```

Bison 文件 parser.y

```
%{
#include "node.h"
#include "sym_entry.h"

extern int yylex();
void yyerror(const char *s) { printf("ERROR: %s\n", s); }
#define YYDEBUG 1

SymTable table;
Node* root;
%}

%define api.value.type {Node*}

%token TTYPE TIDENTIFIER TINTEGER
%token TIF TTHEN TELSE TWHILE TDO
%token TCEQ TCNE TCLT TCLE TCGT TCGE TEQUAL
%token TLPAREN TRPAREN TSQUOTE TDOT
%token TPLUS TMINUS TMUL TDIV TSEMI

%start p

%%

p : d s { root = new Node(P); $$ = root; $$->children.push_back($1); $$->children.push_back($2); }
  ;

d : { $$ = nullptr; }
  | TTYPE TIDENTIFIER TSEMI d {
                          $$ = new Node(D);
                          $2->idx = push_symbol(table,{$2->text, $1->text});
                          $$->children.push_back($1); $$->children.push_back($2); $$->children.push_back($3);
                if ($4) $$->children.push_back($4);
                       }
  ;


s : TIDENTIFIER TEQUAL e TSEMI {
                            $$ = new Node(S);
                            $$->children.push_back($1); $$->children.push_back($2);
                            $$->children.push_back($3); $$->children.push_back($4);
                          }
```

```
43.   | TIF TLPAREN c TRPAREN s {
44.                     $$ = new Node(S);
45.                     $$->children.push_back($1); $$->children.push_back($2);
46.                     $$->children.push_back($3); $$->children.push_back($4); $$->children.pus
    h_back($5);
47.                       }
48.   | TIF TLPAREN c TRPAREN s TELSE s {
49.                             $$ = new Node(S); $$->children.push_back($1);
50.                             $$->children.push_back($2); $$->children.push_back($3); $$->chil
    dren.push_back($4);
51.                             $$->children.push_back($5); $$->children.push_back($6); $$->chil
    dren.push_back($7);
52.                           }
53.   | TWHILE TLPAREN c TRPAREN s {
54.                             $$ = new Node(S);
55.                             $$->children.push_back($1); $$->children.push_back($2);
56.                             $$->children.push_back($3); $$->children.push_back($4); $$->child
    ren.push_back($5);
57.                           }
58.   | s s { $$ = $1; $$->children.push_back($2); }
59.   ;
60.
61. c : e TCGT e { $$ = new Node(C); $$->children.push_back($1); $$->children.push_back($2); $
    $->children.push_back($3); }
62.   | e TCLT e { $$ = new Node(C); $$->children.push_back($1); $$->children.push_back($2); $
    $->children.push_back($3); }
63.   | e TCEQ e { $$ = new Node(C); $$->children.push_back($1); $$->children.push_back($2); $
    $->children.push_back($3); }
64.   ;
65.
66. e : e TPLUS t { $$ = new Node(E); $$->children.push_back($1); $$->children.push_back($2);
    $$->children.push_back($3); }
67.   | e TMINUS t { $$ = new Node(E); $$->children.push_back($1); $$->children.push_back($2);
     $$->children.push_back($3); }
68.   | t { $$ = new Node(E); $$->children.push_back($1); }
69.   ;
70.
71. t : f { $$ = new Node(T); $$->children.push_back($1); }
72.   | t TMUL f { $$ = new Node(T); $$->children.push_back($1); $$->children.push_back($2); $
    $->children.push_back($3); }
73.   | t TDIV f { $$ = new Node(T); $$->children.push_back($1); $$->children.push_back($2); $
    $->children.push_back($3); }
74.   ;
75.
76. f : TLPAREN e TRPAREN {
77.                             $$ = new Node(F);
```

```
78.                         $$->children.push_back($1); $$->children.push_back($2); $$->childr
   en.push_back($3);
79.                       }
80.   | TIDENTIFIER { $$ = new Node(F); $$->children.push_back($1); }
81.   | TINTEGER { $$ = new Node(F); $$->children.push_back($1); }
82.   ;
83.
84. %%
```

（4） 实验结果

测试用例

```
int a;
int b;
int c;
a=2;
b=1;
if (a>b)
  c=a+b;
else
  c=a-b;
```

程序输出

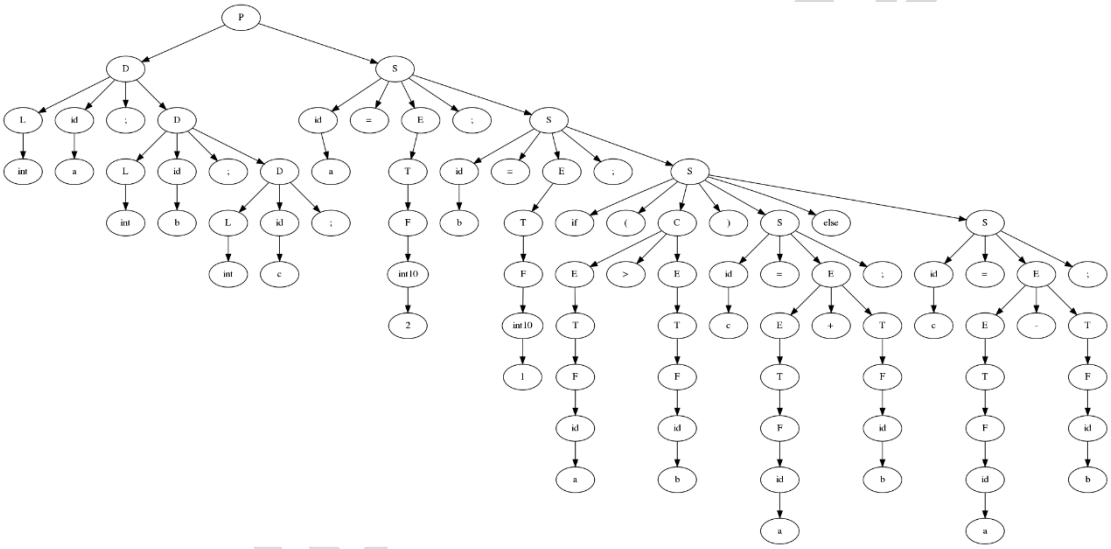


图 2.2 测试用例 2 程序输出

测试用例

```
1.  int var1;
2.  int var2;
3.  float var3;
4.
5.  if (var1 < var3)
6.    while ( (var2 / var3) < 13)
7.      var2 = var1 + var3;
8.  if (var3 > var2)
9.    var2 = var1 / 19;
10. else
11.   var1 = 1;
12. while ( var2 == var3 )
13.   var3 = 107 * var1;
```

程序输出

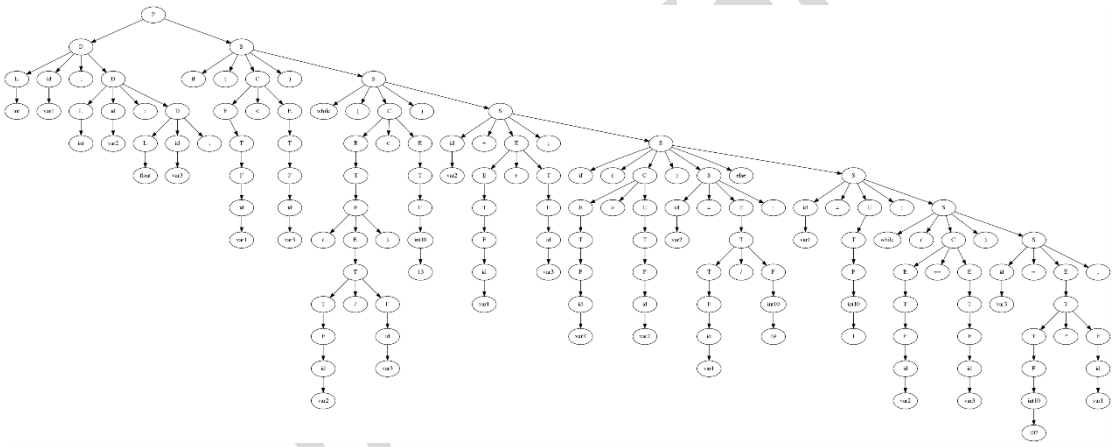


图 2.2　测试用例 2 程序输出

（5）　实验总结

通过学习 Flex 和 Bison 的相关知识，我了解到了如何在实际场景下实现语法分析。Bison 的语法定义和语法动作表示都和课堂所介绍的表示方式类似，因此可以很好的实现知识迁移，将课堂中学到的理论知识转移到实际开发中来，从而加深对语法分析过程的理解。

通过把 AST 树转换为 Dot 文件再用可视化工具显示出来，可以很直观的看到语法分析的结果。

实验三　语义分析及中间代码生成

一 实验目的

通过上机实习，加深对语法制导翻译原理的理解，掌握将语法分析所识别的语法范畴变换为某种中间代码的语义翻译方法。

二 实验内容

实现简单的高级语言源程序的语义处理过程。

三 实验要求

（一） 程序设计要求

（1） 目 标 机：8086 及其兼容处理器

（2） 中间代码：三地址码

（3） 主要数据结构：三地址码表、符号表

（4） 语义分析内容要求：

1) 变量说明语句

2) 赋值语句

3) 控制语句(任选一种)

（5） 其它要求：

1) 将词法分析（扫描器）作为子程序，供语法、语义程序调用；

2) 使用语法制导的语义翻译方法；

3) 编程语言自定；

4) 提供源程序输入界面；

5) 目标代码生成暂不做；

6) 编译后，可查看 TOKEN 串、符号表、三地址码表；

7) 主要数据结构：产生式表、符号表、三地址码表。

所用文法使用实验 2 中的文法。

附录：语义分析源程序范例

```
int a;
int b;
int c;
a=2;
b=1;
if （a>b）
   c=a+b;
else
   c=a-b;
```

（二）实验报告撰写要求

（1）系统功能分析与设计（包括各个子功能模块的功能说明）；

（2）开发平台（开发软硬件环境）；

（3）语义翻译中使用的数据结构；

（4）程序具体设计实现过程（包括主要功能模块的具体实现）。

四 实验总结

（1） 系统功能分析设计

本实验使用 FLEX 结合 Bison 实现。Flex 是 Lex 的更新版本，Bison 是 Yacc 的更新版本。新旧版本语法相似。

本实验主要实现流程为：

1. 编写 lex 文件，进而生成词法分析器 lexer.cc
2. 编写 Bison 文件 parser.y，可以得到语法分析器文件 parser.h 和 parser.cc 文件。
3. 编写具有语义信息的 AST 节点类
4. 实现回填翻译技术。

由于本实验需要使用到语义分析。为了让语法变量能够携带多种信息，我为每种非终结符都设计了对应的 AST 节点类。这些节点类有着与语法类似的继承关系。

回填翻译是本实验的难点，我采用了类似于课件中所展示的方法，最终实现了 IF 语句和 WHILE 语句的回填翻译。

在本实验的实现过程当中，代码生成的核心之处在于保留代码生成的上下文，以及回填技术的实现。其中需要等待回填的标号被存入 AST 节点中。在语义分析的过程中逐步完成对这些数据的回填操作。

本实验完整代码可以在我的 Github 主页查看

https://github.com/YanzheL/hit-compiler-lab

（2） 开发平台

操作系统：Ubuntu 18.04

开发语言：C++ 11

依赖库：Flex, Bison

（3） 语义翻译中使用的数据结构

AST 树节点

```
1. /**
2.  * 表达式类型
3.  */
4. class NExpr : public Serializable {
5.  public:
6.   std::string ToString() override { return {}; };
7. };
8.
9. /**
10.  * 标识符类型
11.  */
12. class NIdentifier : public NExpr {
13.  public:
14.   explicit NIdentifier(std::string &name) : name(std::move(name)) {}
15.  public:
16.   std::string name;
17.   std::string ToString() override { return name; }
18. };
19.
20. /**
21.  * 常整数类型
22.  */
23. class NConstInteger : public NExpr {
24.  public:
25.   long long value;
26.   explicit NConstInteger(long long value) : value(value) {}
27.   std::string ToString() override { return std::to_string(value); }
28. };
29.
30. /**
31.  * 二元算术运算语句类型
32.  */
33. class NBinaryArithOpExpr : public NExpr {
34.  public:
35.   std::string op;
36.   NExpr &lhs;
37.   NExpr &rhs;
38.   NBinaryArithOpExpr(NExpr &lhs, std::string op, NExpr &rhs) : lhs(lhs), rhs(rhs), op(op)
    {}
39.   std::string ToString() override;
40. };
41.
```

```
42. /**
43.  * 可执行语句类型
44.  */
45. class NExecStmt : public Serializable {
46.  public:
47.   std::vector<int> nextlist;
48.   std::string ToString() override { return {}; };
49. };
50.
51. /**
52.  * 二元逻辑表达式
53.  */
54. class NBinaryLogicExpr : public NExpr {
55.  public:
56.   std::vector<int> truelist;
57.   std::vector<int> falselist;
58.   std::string op;
59.   NExpr &lhs;
60.   NExpr &rhs;
61.   NBinaryLogicExpr(NExpr &lhs, std::string op, NExpr &rhs) : lhs(lhs), rhs(rhs), op(op) {}

62.   std::string ToString() override;
63. };
64.
65. /**
66.  * 赋值语句类型
67.  */
68. class NAssignStmt : public NExecStmt {
69.  public:
70.   NIdentifier &lhs;
71.   NExpr &rhs;
72.   NAssignStmt(NIdentifier &lhs, NExpr &rhs) : lhs(lhs), rhs(rhs) {}
73. };
74.
75. /**
76.  * 声明语句类型
77.  */
78. class NDeclStmt : public Serializable {
79.  public:
80.   NDeclStmt(const NIdentifier &type, NIdentifier &id) : type(type), id(id) {}
81.   std::string ToString() override { return {}; };
82.  public:
83.   const NIdentifier &type;
84.   NIdentifier &id;
85. };
86.
```

```
87. /**
88.  * IF 语句类型
89.  */
90. class NIfStmt : public NExecStmt {
91.  public:
92.   explicit NIfStmt(NBinaryLogicExpr &condition, NExecStmt &s1) : condition(condition), s1(
    s1), s2(nullptr) {}
93.  public:
94.   NExecStmt &s1;
95.   NExecStmt *s2;
96.   NBinaryLogicExpr &condition;
97. };
98.
99. /**
100.  * WHILE 语句类型
101.  */
102. class NWhileStmt : public NExecStmt {
103.  public:
104.   explicit NWhileStmt(NBinaryLogicExpr &condition, NExecStmt &s) : condition(condition),
     s(s) {}
105.  public:
106.   NBinaryLogicExpr &condition;
107.   NExecStmt &s;
108. };
109.
110. /**
111.  * 代表整个程序
112.  */
113. class NProgram {
114.  public:
115.   NProgram(std::vector<NDeclStmt *> &declStmts, std::vector<NExecStmt *> &execStmts)
116.       : declStmts(declStmts), execStmts(execStmts) {}
117.  public:
118.   std::vector<NDeclStmt *> &declStmts;
119.   std::vector<NExecStmt *> &execStmts;
120. };
```

符号表

```cpp
class SymInfo : public Serializable {
 public:
  std::string name;
  std::string type;
  int width;
  SymInfo(std::string name, std::string type) : name{name}, type{type} {
    if (type == "int" || type == "float" || type == "long")
      width = 4;
    else if (type == "double") width = 8;
  }
  std::string ToString() override;
};

class SymTable : public Serializable {
 public:
  SymTable() = default;
  virtual ~SymTable() = default;
  virtual int insert(std::string name, std::string type) = 0;
  virtual SymInfo *lookup(int id) = 0;
};

class SimpleSymTable : public SymTable {
 public:
  SimpleSymTable() = default;
  int insert(std::string name, std::string type) override {
    for (const auto &entry:sym_table_)
      if (entry.name == name)
        return -1; // 说明有重复符号
    sym_table_.emplace_back(name, type);
    return sym_table_.size() - 1;
  }
  SymInfo *lookup(int id) override {
    if (id >= sym_table_.size()) return nullptr;
    return &sym_table_[id];
  }
  std::string ToString() override;
 private:
  std::vector<SymInfo> sym_table_;
};
```

（4） 程序具体设计实现过程（包括主要功能模块的具体实现）

本实验所有代码都为本人原创，独立完成。

Flex 文件 lexer.l

```
%option noyywrap
%{
#include <iostream>
#include <string>
#include "node.h"
#include "parser.h"
#define SAVE_TOKEN(t) { yylval.text = new std::string(yytext, yyleng); }
#define TOKEN(t) { yylval.token = t; }
%}
%%
[ \t\n]                 ;
if                      SAVE_TOKEN(TIF) return TIF;
then                    SAVE_TOKEN(TTHEN) return TTHEN;
else                    SAVE_TOKEN(TELSE) return TELSE;
while                   SAVE_TOKEN(TWHILE) return TWHILE;
do                      SAVE_TOKEN(TDO) return TDO;
[a-zA-Z_][a-zA-Z0-9_]*  SAVE_TOKEN(TIDENTIFIER) return TIDENTIFIER;
[0-9]+                  SAVE_TOKEN(TINTEGER) return TINTEGER;
"="                     SAVE_TOKEN(TEQUAL) return TEQUAL;
"=="                    SAVE_TOKEN(TCEQ) return TCEQ;
"!="                    SAVE_TOKEN(TCNE) return TCNE;
"<"                     SAVE_TOKEN(TCLT) return TCLT;
"<="                    SAVE_TOKEN(TCLE) return TCLE;
">"                     SAVE_TOKEN(TCGT) return TCGT;
">="                    SAVE_TOKEN(TCGE) return TCGE;
"("                     SAVE_TOKEN(TLPAREN) return TLPAREN;
")"                     SAVE_TOKEN(TRPAREN) return TRPAREN;
"."                     SAVE_TOKEN(TDOT) return TDOT;
"'"                     SAVE_TOKEN(TSQUOTE) return TSQUOTE;
"+"                     SAVE_TOKEN(TPLUS) return TPLUS;
"-"                     SAVE_TOKEN(TMINUS) return TMINUS;
"*"                     SAVE_TOKEN(TMUL) return TMUL;
"/"                     SAVE_TOKEN(TDIV) return TDIV;
";"                     SAVE_TOKEN(TSEMI) return TSEMI;
.                       std::cout<< "Unknown token!" << std::endl; yyterminate();
%%
```

Bison 文件 parser.y

```
%{
#include "node.h"
#include "sym_table.h"
#include "codegen.h"
#include <iostream>

extern int yylex();
SymTable* table = new SimpleSymTable;
Generator context;
void yyerror(const char *s) { printf("ERROR: %s\n", s); }
#define YYDEBUG 1

NProgram *program;
%}

%union {
    NProgram *program;
    NExpr *expr;
    NIdentifier *ident;
    NConstInteger *num;
    NBinaryArithOpExpr *arithexpr;
    NBinaryLogicExpr *logicexpr;
    NDeclStmt *declstmt;
    NExecStmt *execstmt;
    NAssignStmt *assignstmt;
    NIfStmt *ifstmt;
    NWhileStmt *whilestmt;

    std::vector<NDeclStmt*> *decls;
    std::vector<NExecStmt*> *stmts;
    std::vector<NExpr*> *exprvec;

    std::string *text;
    int token;
    int instr;
    std::vector<int> *labellist;
}

%token <text> TIDENTIFIER TINTEGER
%token <text> TIF TTHEN TELSE TWHILE TDO
%token <text> TCEQ TCNE TCLT TCLE TCGT TCGE TEQUAL
%token <text> TLPAREN TRPAREN TSQUOTE TDOT
%token <text> TPLUS TMINUS TMUL TDIV TSEMI

```

```
45.
46. %type <text> comparison arith_op1 arith_op2
47. %type <expr> expr
48. %type <logicexpr> logicexpr
49. %type <arithexpr> arithexpr
50. %type <ident> ident
51. %type <declstmt> decl
52. %type <execstmt> stmt
53. %type <ifstmt> if_stmt if_else_stmt
54. %type <whilestmt> while_stmt
55. %type <assignstmt> assign_stmt
56. %type <stmts> stmts
57. %type <decls> decls
58. %type <num> const_int
59. %type <program> program
60. %type <instr> M
61. %type <labellist> N
62.
63. %start program
64.
65. %%
66.
67. program : decls stmts { program = new NProgram(*$1, *$2); $$ = program; }
68.   ;
69.
70. decls : decl { $$ = new std::vector<NDeclStmt *>(); $$->push_back($1); }
71.   | decls decl { $$ = $1; $$->push_back($2); }
72.   ;
73.
74. stmts : stmt { $$ = new std::vector<NExecStmt *>(); $$->push_back($1); }
75.   | stmts stmt { $$ = $1; $$->push_back($2); }
76.   ;
77.
78. ident : TIDENTIFIER { $$ = new NIdentifier(*$1); delete $1; }
79.   ;
80.
81. const_int : TINTEGER { $$ = new NConstInteger(std::stoi(*$1)); delete $1; }
82.   ;
83.
84. comparison : TCGT | TCLT | TCEQ
85.   ;
86.
87. arith_op1 : TPLUS | TMINUS
88.   ;
89.
90. arith_op2 : TMUL | TDIV
```

```
91.   ;
92.
93. decl : ident ident TSEMI {
94.     $$ = new NDeclStmt(*$1, *$2);
95.     table->insert($2->name, $1->name);
96.   }
97.   ;
98.
99. if_stmt : TIF TLPAREN logicexpr TRPAREN M stmt {
100.     $$ = new NIfStmt(*$3, *$6);
101.     context.BackPatch($3->truelist, $5);
102.     context.Merge($$->nextlist, $3->falselist);
103.     context.Merge($$->nextlist, $6->nextlist);
104.   }
105.   ;
106.
107. if_else_stmt : if_stmt N TELSE M stmt {
108.     $$ = $1; $$->s2 = $5;
109.     context.BackPatch($1->condition.falselist, $4);
110.     $$->nextlist.clear();
111.     context.Merge($$->nextlist, *$2);
112.     context.Merge($$->nextlist, $5->nextlist);
113.     context.Merge($$->nextlist, $$->s1.nextlist);
114.   }
115.   ;
116.
117. M : { $$ = context.nextinstr; }
118.   ;
119.
120. N : { $$ = new std::vector<int>{context.nextinstr}; context.Gen(new CodeLine({"goto","_"
    }, 1)); }
121.   ;
122.
123. while_stmt : TWHILE M TLPAREN logicexpr TRPAREN M stmt {
124.     $$ = new NWhileStmt(*$4, *$7);
125.     context.BackPatch($7->nextlist, $2);
126.     context.BackPatch($4->truelist, $6);
127.     $$->nextlist = $4->falselist;
128.     context.Gen(new CodeLine({"goto",std::to_string($2)}));
129.   }
130.   ;
131.
132. assign_stmt : ident TEQUAL expr TSEMI {
133.     $$ = new NAssignStmt(*$1, *$3);
134.     context.Gen(new CodeLine({"t", "=", $3->ToString()}));
135.     context.Gen(new CodeLine({$1->ToString(), "=", "t"}));
```

```
136.   }
137.   ;
138.
139. stmt : assign_stmt
140.   | if_stmt
141.   | if_else_stmt
142.   | while_stmt
143.   ;
144.
145. logicexpr : expr comparison expr {
146.     $$ = new NBinaryLogicExpr(*$1, *$2, *$3);
147.     $$->truelist.push_back(context.nextinstr);
148.     context.Gen(new CodeLine({"if", $1->ToString(), *$2, $3->ToString(), "goto", "_"}, 5
    ));
149.     $$->falselist.push_back(context.nextinstr);
150.     context.Gen(new CodeLine({"goto", "_"}, 1));
151.   }
152.   ;
153.
154. arithexpr : expr arith_op2 expr { $$ = new NBinaryArithOpExpr(*$1, *$2, *$3); }
155.   | expr arith_op1 expr { $$ = new NBinaryArithOpExpr(*$1, *$2, *$3); }
156.   ;
157.
158. expr :  ident { $<ident>$ = $1; }
159.   | const_int
160.   | arithexpr
161.   | logicexpr
162.   | TLPAREN expr TRPAREN { $$ = $2; }
163.   ;
164.
165. %%
```

代码生成器类

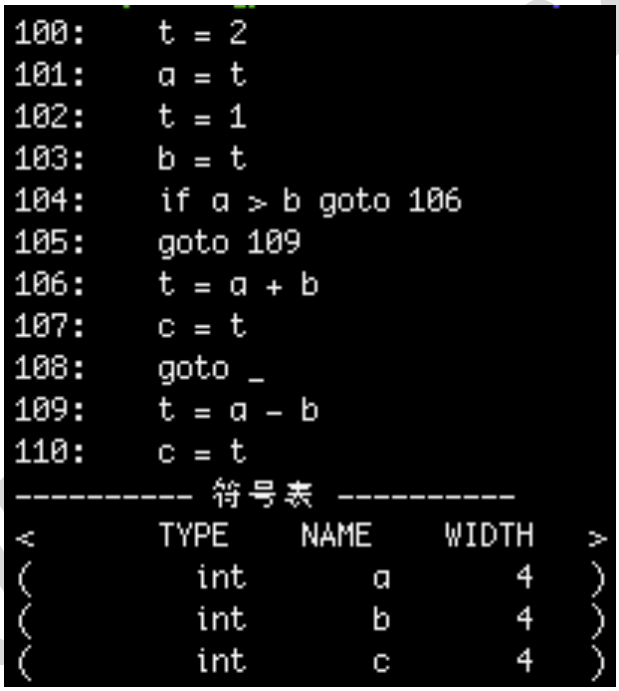
```
1.  class CodeLine {
2.   public:
3.    explicit CodeLine(std::vector<std::string> vars) : vars(std::move(vars)), unknown_idx(-
    1) {}
4.    CodeLine(std::vector<std::string> vars, int unknownIdx) : vars(std::move(vars)), unknown
    _idx(unknownIdx) {}
5.    std::string ToString() {
6.      std::stringstream ss;
7.      for (const auto &var:vars)
8.        ss << var << " ";
9.      return ss.str();
10.   }
11.  public:
12.   std::vector<std::string> vars;
13.   int unknown_idx;
14. };
15.
16. class Generator : public Serializable {
17.  public:
18.   const int START_ADDR = 100;
19.   int nextinstr = START_ADDR;
20.   std::vector<CodeLine *> codes;
21.   Generator() = default;
22.   int Gen(CodeLine *expr) {
23.     codes.push_back(expr);
24.     return ++nextinstr;
25.   }
26.   void BackPatch(std::vector<int> &list, int label) {
27.     for (int line:list) {
28.       assert(line < nextinstr);
29.       CodeLine *expr = codes[line - START_ADDR];
30.       assert(expr->unknown_idx != -1);
31.       expr->vars[expr->unknown_idx] = std::to_string(label);
32.     }
33.   }
34.   static void Merge(std::vector<int> &dst, const std::vector<int> &src) {
35.     dst.insert(dst.end(), src.begin(), src.end());
36.   }
37.   std::string ToString() override;
38. };
```

（5） 实验结果

测试用例

```
int a;
int b;
int c;
a=2;
b=1;
if (a>b)
  c=a+b;
else
  c=a-b;
```

程序输出

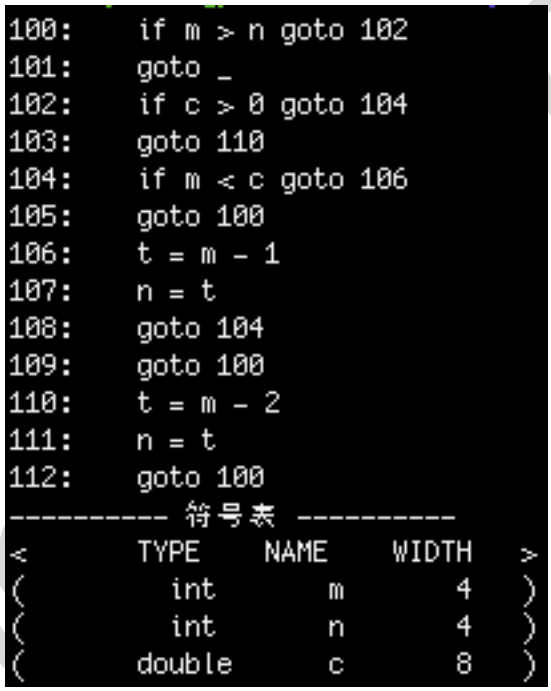


测试用例

```
int m;
int n;
double c;

while (m>n)
  if (c>0)
    while (m<c)
      n = m - 1;
  else
    n = m - 2;
```

程序输出



（6） 实验总结

本实验的成功实现让我对中间代码生成的这个知识点有了更加深入的理解，特别是回填翻译技术，以及三地址码的机内表示等等。

但我的实现过程还是有一些不足之处。例如，对于赋值语句使用的临时变量，我每次用的都是 t 来表示临时变量，而不是每次都分配一个新的临时变量。还有一点则是，如果赋值语句用到的是常量，如 a=1，我也会生成一个不必要的临时变量来存放 1。这个问题需要今后继续改进。