浙江大学

sysY 语言编译器的实现

组员 1：王子骏 3200103994

组员 2：陈绍文 3200102333

组员 3：胡梓敬 3200105635

专业：计算机科学与技术

课程：编译原理

授课教师：李莹

一、介绍

1.1 sysY介绍

本次实验完成了一个sysY语言的编译器，能够分析sysY语言的语法，并将其编译至LLVM IR，最后再编译至目标代码(.o 文件)。

sysY 是一种简单的编程语言，通常用于教学和学术研究目的。sysY 语言的设计旨在提供一个易于理解和实现的语言，用于介绍编译器设计和实现的基本概念。

我们实现的sysY编译器支持以下sysY语言特性:

1. 静态类型：sysY 是一种静态类型语言，所有变量在编译时必须声明其类型，并且类型检查是在编译阶段进行的。类型检查可以帮助捕捉许多常见的编程错误。
2. 基本数据类型：sysY 支持整数和浮点数类型。整数类型可以包括有符号和无符号的整数，可以指定不同的位宽。浮点数类型可以是单精度或双精度。
3. 数组：sysY 支持一维和多维数组。数组的长度可以在声明时指定或在运行时动态分配。可以使用索引访问和修改数组元素。
4. 表达式：sysY 支持基本的算术和逻辑表达式，包括算术运算符（如加法、减法、乘法、除法）、比较运算符（如等于、不等于、大于、小于）和逻辑运算符（如与、或、非）。
5. 控制流语句：sysY 支持条件语句和循环语句。条件语句包括 if-else 语句，根据条件的结果选择不同的执行路径。循环语句包括 for 循环和 while 循环，用于重复执行一段代码块。
6. 函数：sysY 支持函数的定义和调用。函数可以接受参数并返回值。函数定义包括函数名称、参数列表和函数体。函数调用可以将实际参数传递给函数，并接收返回值。
7. 输出语句：sysY 提供了输出语句，用于在程序执行过程中打印输出结果。可以打印字符串常量和变量的值。
8. 作用域：sysY 使用块作用域。变量在定义它们的代码块内可见，并且可以被内部的代码块重新定义。
9. 没有指针和复杂数据结构：sysY 简化了语言设计，不支持指针和复杂的数据结构（如结构体和类）。它专注于基本的数据类型和控制结构。

1.2 运行环境说明

- Flex & Bison：使用flex和bison生成词法分析器和语法分析器。
- LLVM-14：基于LLVM标准完成AST的构建、IR代码的生成；同时使用LLVM来完成编译器优化、目标代码生成。
- CMake：使用CMake来构建工程。

1.2.1 安装参考

1. 安装环境：Ubuntu22.04
2. Flex : sudo apt-get install flex
3. Bison : sudo apt-get install Bison
4. LLVM : sudo apt-get install llvm-14 (由于llvm不同版本接口有所变化，在实验中llvm版本在11即以下无法进行工程构建，请尽量和该版本保持一致)

1.2.2 工程搭建

1. 进入工程根目录，工程结构如下：

```
docs/
io/
lib/
out/
Ref/
simple_demo/
src/
tests/
scrips/
README.md
git_log.txt
CmakeLists.txt
```

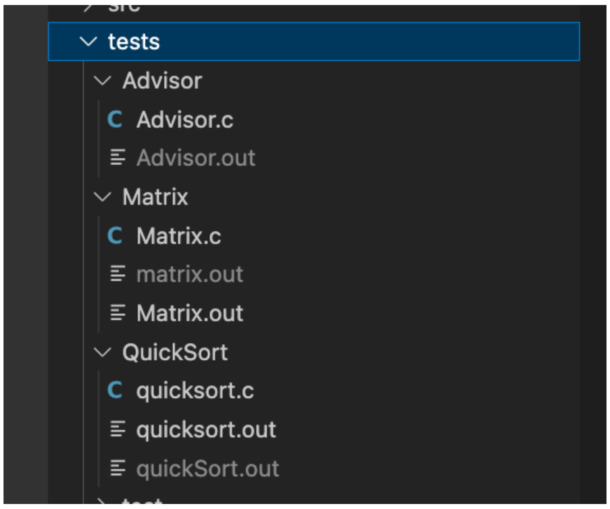
2. 用CMake搭建工程，在 Compiler 文件夹下进行以下命令

```
cd Compiler
bash ./scripts/cmake_srcs.sh
```

以上命令在Complier/build 目录编译得到可执行文件 Compiler

1.2.3 使用说明

测试代码在 tests 文件夹下



测试指令在 scrips 文件夹下

Complier/build 目录下有编译得到的可执行文件 Compiler

执行以下指令:

```
cd Compiler
bash ./scripts/Quicksort_run.sh
bash ./scripts/Matrix_run.sh
bash ./scripts/Advisor_run.sh
```

以上三个sh文件分别测试三个测试点

1.3 代码规范

详细的代码规范请见 docs/styleGuide.md 。

1.4 分工说明

二、实验设计文档

2.1 词法分析设计

在实验报告的词法分析部分，我们将详细介绍sysY语言的词法规范和词法分析器的设计与实现过程。我们将描述sysY语言中各种词法单元的定义和识别规则，以及词法分析器的工作原理和算法。我们还将讨论词法错误处理和测试验证的方法，以确保词法分析器的正确性和鲁棒性。通过深入了解词法分析的重要性和功能，我们可以更好地理解编译器的工作原理和实现过程。

2.1.1 词法单元的定义和识别规则

1. 关键字（Keywords）:
    - 控制结构关键字:
        - if： 用于条件语句，根据条件表达式的结果执行相应的代码块。
        - else： 在条件语句中可选的分支，用于处理条件不满足的情况。
        - while： 用于循环语句，当条件满足时重复执行代码块。
    - 数据类型关键字:
        - int： 表示整数类型。
        - float： 表示浮点数类型。
        - char： 表示字符类型。
    - 修饰符关键字:
        - const： 用于声明常量，表示该变量的值在程序执行期间不可修改。

        - void ： 表示无返回值或不接受参数的函数。

2. 标识符（Identifiers）： 标识符用来表示变量、函数、类等用户自定义的命名。
    - 标识符由字母、数字和下划线组成。
    - 标识符必须以字母或下划线开头，不能以数字开头。
    - 标识符区分大小写，因此大小写不同的标识符被视为不同的符号。
3. 常量（Literals）： 常量是固定的数值或字符值，包括整数、浮点数、字符、字符串等。
    - 整数常量：整数常量是不带小数点的数值，可以使用十进制、八进制或十六进制表示。例如： `42` 、 `0b1010` 、 `0o77` 、 `0xFF` 。
    - 浮点数常量：浮点数常量是带有小数点的数值。可以使用十进制表示，也可以使用科学计数法表示。例如： `3.14` 、 `1.23e-5` 。
    - 字符常量：字符常量是单个字符，用单引号括起来。例如： `'a'` 、 `'b'` 、 `'%'` 。
    - 字符串常量：字符串常量是由多个字符组成的序列，用双引号括起来。例如： `"Hello, world!"` 、 `"sysY"` 。
4. 运算符（Operators）： 运算符用于执行各种操作，如算术运算、逻辑运算、比较运算等。sysY语言中包括常见的运算符，如加法、减法、乘法、除法、赋值、逻辑与、逻辑或等。
    - 算术运算符：用于执行基本的算术操作，如加法、减法、乘法和除法。常见的算术运算符包括加号 `+` 、减号 `-` 、乘号 `*` 和除号 `/` 。
    - 赋值运算符：用于将一个值赋给变量。赋值运算符使用等号 `=` 表示，例如 `a = 5` 。
    - 逻辑运算符：用于执行逻辑操作，如逻辑与、逻辑或和逻辑非。逻辑与运算符表示为双与号 `&&` ，逻辑或运算符表示为双竖线 `||` ，逻辑非运算符表示为感叹号 `!` 。
    - 比较运算符：用于比较两个值之间的关系，如相等、不相等、大于、小于等。常见的比较运算符包括等于号 `==` 、不等于号 `!=` 、大于号 `>` 、小于号 `<` 、大于等于号 `>=` 和小于等于号 `<=` 。
    - 位运算符：用于对二进制位进行操作。常见的位运算符包括按位与 `&` 、按位或 `|` 、按位取反 `~` 、按位异或 `^` 、左移 `<<` 和右移 `>>` 。
    - 条件运算符：也称为三元运算符，用于根据条件选择不同的值。条件运算符使用问号 `?` 和冒号 `:` 组合，例如 `condition ? value1 : value2` 。
    - 其他运算符：sysY语言还支持其他一些运算符，如取地址运算符 `&` 、取值运算符 `*` 、下标运算符 `[]` 等。

    运算符在词法分析阶段会被识别为独立的词法单元，并传递给语法分析器进行解析。不同的运算符具有不同的优先级和结合性，编译器会按照运算符的规则进行解析和计算，确保表达式的正确性和准确性。

2.1.2 词法规范：正则表达式

在Lex中，正则表达式用于描述词法单元的模式。它们指定了需要匹配的字符序列的规则，并可以与特定的动作关联。以下是一些Lex约定用于描述正则表达式的常见语法：

格式	含义
a	字符a
"a"	即使a是一个元字符，它仍是字符a
\a	即使a是一个元字符，它仍是字符a
a*	a的零次或多次重复
a+	a的一次或多次重复
a?	一个可选的a
a|b	a或b
(a)	a本身
[abc]	字符a、b或c中的任一个
[a-d]	字符a、b、c或d中的任一个
[^ab]	除了a或b外的任一个字符
.	除了新行之外的任一个字符
{xxx}	名字xxx表示的正则表达式

2.1.3 词法分析器设计

在词法分析器设计中，采用词法分析器生成工具（Flex）是一种常见的选择。Flex是一个词法分析器生成器，它可以根据给定的正则表达式规则和相应的动作生成用于词法分析的自动识别器。

Lex输入文件由三个部分组成：定义（defination）集，规则（rule）集以及辅助程序（auxiliary）集或用户程序（user routine）集。

```
{definations}
%%
{rules}
%%
{auxiliary routines}
```

2.1.3.1 定义部分

```
%{
#include "AST.hpp"
#include "parser.hpp"
#include <stdio.h>
#include <iostream>
#include <string>

#define SAVE_TOKEN yylval.strVal = new std::string(yytext, yyleng)
// 因为 Flex 会用到 Bison 中关于 token 的定义
// 所以需要 include Bison 生成的头文件
extern "C" int yywrap() {return 1;}

using namespace std;

%}
```

2.1.3.2 规则部分

```
/* 空白符和注释 */
WHTIESPACE          [ \t\n\r]*
LINECOMMET           "//".*
BLOCKCOMMET           "/*"[^*]*[*]+([^*/][^*]*[*]+)*"/"

/* 整数字面量 */
DECIMAL            [1-9][0-9]*
OCTAL            0[0-7]*
HEXADECIMAL          0[xX][0-9a-fA-F]+
/* 浮点数 */
REAL                [0-9]+\.[0-9]+

/* 字符串 */
STRING               \"(\\.|[^\"\\])*?\"

/* 标识符 */
IDENTIFIER           [a-zA-Z_][a-zA-Z0-9_]*

%%

   /* 空白符和注释 */
{WHTIESPACE}          {;}
{LINECOMMET}           {;}
{BLOCKCOMMET}          {;}

   /* 运算符 */
   /* 关系运算符 */
"=="                  {return EQU;}
"!="                  {return NEQ;}
"<"                   {return LES;}
"<="                  {return LEQ;}
">"                   {return GRE;}
">="                  {return GEQ;}

   /* 算术运算符 */
"+"                   {return ADD;}
"-"                   {return SUB;}
"*"                   {return MUL;}
"/"                   {return DIV;}
"%"                   {return MOD;}

   /* 逻辑运算符 */
"&&"                  {return AND;}
"||"                  {return OR;}
"!"                   {return NOT;}

   /* 位运算符 */
```

```
"&"                 {return BAND;}
"|"                {return BOR;}
"^"                 {return BXOR;}

  /*分隔符*/
"("              { return LPAREN; }
")"              { return RPAREN; }
"{"              { return LBRACE; }
"}"              { return RBRACE; }
"["         {return LBRACKET;}
"]"         {return RBRACKET;}
","                {return COMMA;}
";"                {return SEMI;}

  /* 赋值运算符 */
"="                 {return ASSIGN;}

  /* 特殊符号 */
"."                {return DOT;}
":"                {return COLON;}
"?"                 {return QUES;}
"..."       {return ELLIPSES;}
"ptr"         {return PTR;}

  /* 关键字 */
  /* 数据类型 */
"int"               {SAVE_TOKEN; return INT;}
"short"               {SAVE_TOKEN; return SHORT;}
"char"                {SAVE_TOKEN; return CHAR;}
"void"                {SAVE_TOKEN; return VOID;}

  /* 跳转结构 */
"return"              {return RETURN;}
"continue"              {SAVE_TOKEN; return CONTINUE;}
"break"               {SAVE_TOKEN; return BREAK;}

  /* 分支结构 */
"if"                {SAVE_TOKEN; return IF;}
"else"                {SAVE_TOKEN; return ELSE;}
```

```
  /* 循环结构 */
"for"              {SAVE_TOKEN; return FOR;}
"while"             {SAVE_TOKEN; return WHILE;}

"static"            {SAVE_TOKEN; return STATIC;}

{STRING}            {
                    yylval.strVal = new std::string("");
                    for (int i = 1; i <= yyleng-2; i++){
                        if (yytext[i] == '\\') {
                            i ++;
                            switch(yytext[i]) {
                                case 'n': yylval.strVal->push_back('\n'); break;
                                case 't': yylval.strVal->push_back('\t'); break;
                                default: yylval.strVal->push_back(yytext[i]);
                            }
                        }
                        else {
                            yylval.strVal->push_back(yytext[i]);
                        }
                    }
                    return CONST_STR;
                }


{IDENTIFIER}          {SAVE_TOKEN; return IDENTIFIER;}



{DECIMAL}             { yylval.intVal = strtol(yytext, nullptr, 0); return CONST_INT; }
{OCTAL}               { yylval.intVal = strtol(yytext, nullptr, 0); return CONST_INT; }
{HEXADECIMAL}         { yylval.intVal = strtol(yytext, nullptr, 0); return CONST_INT; }

{REAL}                { yylval.floatVal = strtol(yytext, nullptr, 0); return CONST_FLOAT; }

"\'"[^\\']"\'"      { yylval.charVal = yytext[1]; return CONST_CHAR; }



.                   { printf("Unknown token!/n");}
```

2.2 语法分析设计

2.2.1 介绍

语法分析在编译器中扮演着关键的角色，它将源代码转化为抽象语法树表示，为后续的编译器阶段提供了基础。

2.2.2 语法规范

以下是sysY语言的正式语法规范，使用巴科斯范式（BNF）描述语法规则。

以下为设计过程中的部分规则描述：

```
<program> ::= <declaration_list>

<declaration_list> ::= <declaration> | <declaration_list> <declaration>

<declaration> ::= <type_specifier> <identifier> ";" | <type_specifier> <identifier> "(" ")"
<compound_statement>

<type_specifier> ::= "int" | "void"

<identifier> ::= <letter> <identifier_tail>

<identifier_tail> ::= <letter_or_digit> <identifier_tail> | ε

<letter> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"

<letter_or_digit> ::= <letter> | <digit>

<compound_statement> ::= "{" <local_declarations> <statement_list> "}"

<local_declarations> ::= <local_declaration> | <local_declarations> <local_declaration>
```

```
<local_declaration> ::= <type_specifier> <identifier> ";"

<statement_list> ::= <statement> | <statement_list> <statement>

<statement> ::= <expression_statement> | <compound_statement> | <selection_statement> |
<iteration_statement> | <return_statement>

<expression_statement> ::= <expression> ";" | ";"

<selection_statement> ::= "if" "(" <expression> ")" <statement> | "if" "(" <expression> ")" <statement> "else"
<statement>

<iteration_statement> ::= "while" "(" <expression> ")" <statement>

<return_statement> ::= "return" <expression> ";" | "return" ";"

<expression> ::= <var> "=" <expression> | <simple_expression>

<var> ::= <identifier>

<simple_expression> ::= <additive_expression> | <simple_expression> <relop> <additive_expression>

<additive_expression> ::= <term> | <additive_expression> <addop> <term>

<term> ::= <factor> | <term> <mulop> <factor>

<factor> ::= <var> | <call> | <num> | "(" <expression> ")"

<call> ::= <identifier> "(" ")" | <identifier> "(" <args> ")"

<args> ::= <expression> | <args> "," <expression>

<relop> ::= "<" | "<=" | ">" | ">=" | "==" | "!="

<addop> ::= "+" | "-"

<mulop> ::= "*" | "/"

<num> ::= <digit> | <digit> <num>
```

```
<digit> ::= "0" | "1" | ... | "9"
```

sysY语言的关键字包括： `int` ，`void` ，`if` ，`else` ，`while` ，`return` 。

sysY语言的运算符包括： `+` ，`-` ，`*` ，`/` ，`=` ，`<` ，`<=` ，`>` ，`>=` ，`==` ，`!=` 。

sysY语言的语法结构包括：程序（ `<program>` ）、声明列表（ `<declaration_list>` ）、声明（ `<declaration>` ）、类型说明符（ `<type_specifier>` ）、标识符（ `<identifier>` ）、复合语句（ `<compound_statement>` ）、局部声明列表（ `<local_declarations>` ）、局部声明（ `<local_declaration>` ）、语句列表（ `<statement_list>` ）、语句（ `<statement>` ）、表达式语句（ `<expression_statement>` ）、选择语句（ `<selection_statement>` ）、循环语句（ `<iteration_statement>` ）、返回语句（ `<return_statement>` ）、表达式（ `<expression>` ）、变量（ `<var>` ）、简单表达式（ `<simple_expression>` ）、加法表达式（ `<additive_expression>` ）、项（ `<term>` ）、因子（ `<factor>` ）、函数调用（ `<call>` ）、实参列表（ `<args>` ）、关系运算符（ `<relop>` ）、加法运算符（ `<addop>` ）、乘法运算符（ `<mulop>` ）、数字（ `<num>` ）、数字字符（ `<digit>` ）。

这些语法规则和符号定义了sysY语言的语法结构和语法规则，编译器在语法分析阶段将根据这些规则进行源代码的解析和构建抽象语法树（AST）。

2.2.3 语法分析器设计方案

在设计sysY编译器的语法分析器时，采用了自底向上的语法分析方法，并选择了LALR(1)解析算法。为了实现语法分析器，使用了Yacc/Bison作为语法分析器生成工具。

自底向上的语法分析方法以输入的源代码作为起点，逐步构建语法规则的右部，并最终推导出起始符号，生成抽象语法树（AST）。相较于自顶向下方法，自底向上方法可以处理更复杂的文法，并具有更强的语法表达能力。

LALR(1)解析算法是一种常用的自底向上解析算法，它使用了向前看一个符号（Look-Ahead 1）来进行决策。LALR(1)算法通过构建LR(1)项目集族和状态转换表来解析输入的源代码。这种解析算法具有较高的效率和广泛的应用性。

为了生成语法分析器，选择了Yacc/Bison作为工具。Yacc/Bison是一对用于生成语法分析器的工具，它们根据提供的文法规范自动生成相应的语法分析器代码。通过定义文法规则和语义动作，Yacc/Bison可以生成解析输入的源代码，并构建抽象语法树。

在sysY编译器的设计中，通过编写sysY语言的文法规范并使用Yacc/Bison工具，生成了相应的语法分析器。该语法分析器利用LALR(1)解析算法来解析输入的源代码，并生成抽象语法树作为后续编译过程的基础。

采用自底向上的语法分析方法、LALR(1)解析算法以及Yacc/Bison工具，可以确保sysY编译器能够准确、高效地进行语法分析，并生成正确的抽象语法树。

2.2.3.1 定义部分

定义部分定义了CFG语法中 non-terminal 的类型、 terminal 的TOKEN和运算符的优先级。定义优先级时， %left ， %right 定义结合性，越后定义的运算优先级高。

```
%union {
    std::string* strVal;
  float floatVal;
    int intVal;
  char charVal;
    BaseAST *astVal;
  CompUnits *compUnits;
  Stmts *stmts;
  Exprs *exprs;
    ArgAST *argVal;
    ArgListAST *argList;
    ExprListAST *exprList;
    ExprAST *expVal;
    PointerType *ptrType;
}

/* 终结符 */
// lexer 返回的所有 token 种类的声明
%token EQU NEQ LES LEQ GRE GEQ
%token ADD SUB MUL DIV MOD
%token AND OR NOT
%token BAND BOR BXOR
%token LPAREN RPAREN LBRACE RBRACE  LBRACKET RBRACKET COMMA SEMI
%token ASSIGN DOT COLON QUES ELLIPSES PTR

%token <strVal> INT CHAR SHORT VOID
```

```
%token RETURN CONTINUE BREAK STATIC
%token IF ELSE
%token FOR WHILE
%token STATIC

%token CONST
%token <strVal> IDENTIFIER
%token <intVal> CONST_INT
%token <charVal> CONST_CHAR
%token <strVal> CONST_STR
%token CONST_FLOAT
// 非终结符的类型定义

%type <astVal> Program
%type <compUnits> CompUnit
%type <astVal> FuncDef
%type <strVal> FuncType

%type <argList> ArgList
%type <argList> _ArgList
%type <argVal> Arg
%type <ptrType> PtrType

%type <exprList> ExpList
%type <exprList> _ExpList

%type <astVal> Decl
%type ConstDecl
%type ConstDef
%type ConstList
%type <astVal> ConstExp
%type ConstInitVal
%type <astVal> VarDecl
%type <strVal> Btype
%type VarList
%type <astVal> VarDef



%type <exprs> ArrDef ArrVal
```

```
%type <astVal> InitVal

%type <astVal> Block
%type <astVal> BlockItem
%type <stmts>  BlockItemNew
%type <astVal> Stmt SmooStmt

%type <astVal> PrimaryExp
%type <astVal> Exp
%type <astVal> ElseState
%type <astVal> RetState

%type <astVal> LVal
%type <intVal> Number
%type <astVal> Constant

%type <astVal> ArrValF

/* 优先级和结合性定义 */
%right  ASSIGN
%left OR
%left AND
%left BOR
%left BXOR
%left BAND
%left EQU NEQ
%left GEQ GRE LES LEQ
%left ADD SUB
%left MUL DIV MOD
```

2.2.3.2 规则部分

```
Program
  : CompUnit                    { $$ = new ProgramAST((CompUnits*)$1); Root = $$;}
  ;

CompUnit
```

```
    : CompUnit STATIC Decl                      { $$ = (CompUnits*)$1; $$->push_back((CompUnitAST*)$3); }
    | CompUnit FuncDef                        { $$ = (CompUnits*)$1; $$->push_back((CompUnitAST*)$2); }
    |                           { $$ = new CompUnits(); }
    ;

/* Decl          ::= ConstDecl | VarDecl; */
Decl
    : ConstDecl
    | VarDecl                                   { $$ = $1; }
    ;

/* ConstDecl     ::= "const" BType ConstDef {"," ConstDef} ";"; */
ConstDecl
    : CONST Btype ConstDef ConstList SEMI
    ;

ConstList
    : ConstList COMMA ConstDef
    |
    ;

/* BType         ::= "int"; */
Btype
    : VOID                          { $$ = $1; }
    | INT                         { $$ = $1; }
    | SHORT                         { $$ = $1; }
    | CHAR                          { $$ = $1; }
    ;

/* ConstDef      ::= IDENT "=" ConstInitVal; */
ConstDef
    : IDENTIFIER ASSIGN ConstInitVal
    ;

/* ConstInitVal  ::= ConstExp; */
ConstInitVal
    : ConstExp
    ;
```

```
/* VarDecl       ::= BType VarDef {"," VarDef} ";"; */
VarDecl
    : Btype VarDef VarList SEMI                    { $$ = new VarDeclAST(*$1, (VarDefAST*)$2);}
  | Btype IDENTIFIER ArrDef SEMI          { $$ = new ArrDefAST(*$1, *$2, (Exprs*)$3); }
    ;

VarList
    : VarList COMMA VarDef
    |                                              { ; }
    ;

/* VarDef       ::= IDENT | IDENT "=" InitVal; */
VarDef
    : IDENTIFIER                                  { $$ = new VarDefAST(*$1);}
    | IDENTIFIER ASSIGN InitVal                    { $$ = new VarDefAST(*$1, (ExprAST*)$3);}
    ;

ArrDef
  : ArrDef LBRACKET ConstExp RBRACKET       { $$ = (Exprs*)$1; $$->push_back((ExprAST*)$3); }
  |                       { $$ = new Exprs(); }
  ;

/* InitVal       ::= Exp; */
InitVal
    : Exp                                  { $$ = $1; }
    ;

FuncDef
    : FuncType IDENTIFIER LPAREN ArgList RPAREN SEMI    { $$ = new FuncDefAST(*$1, *$2,
(ArgListAST*)$4); }
    | FuncType IDENTIFIER LPAREN ArgList RPAREN Block   { $$ = new FuncDefAST(*$1, *$2,
(ArgListAST*)$4, (BlockAST*)$6); }
    ;



/* FuncType      ::= "void" | "int"; */
FuncType
    : INT                                  { $$ = $1; }
    | VOID                                   { $$ = $1; }
    | CHAR                          { $$ = $1; }
    ;


ArgList
    : _ArgList COMMA Arg                { $$ = $1; $$->push_back($3);  }
  | _ArgList COMMA ELLIPSES             { $$ = $1; $$->SetVarArg();  }
  | Arg                     { $$ = new ArgListAST(); $$->push_back($1);  }
  | ELLIPSES                    { $$ = new ArgListAST(); $$->SetVarArg();  }
  |                       { $$ = new ArgListAST(); }
  ;

_ArgList
    : _ArgList COMMA Arg                { $$ = $1; $$->push_back($3);  }
  | Arg                     { $$ = new ArgListAST(); $$->push_back($1);  }
  ;

PtrType
    : Btype PTR                                 { $$ = new PointerType(*$1); }
    ;

Arg
    : Btype IDENTIFIER                  { $$ = new ArgAST(*$1, *$2);  }
  | PtrType IDENTIFIER                  { $$ = new ArgAST($1, *$2);  }
  | PtrType                         { $$ = new ArgAST($1);  }


/* Block        ::= "{" {BlockItem} "}"; */
Block
    : LBRACE BlockItemNew RBRACE              { $$ = new BlockAST((Stmts*)$2);}
    ;

BlockItemNew
    : BlockItemNew BlockItem          { $$ = (Stmts*)$1; if ($2 != NULL)$$->push_back((CompUnitAST*)$2); }
    |                       { $$ = new Stmts(); }
    ;

/* BlockItem     ::= Decl | Stmt; */
BlockItem
```

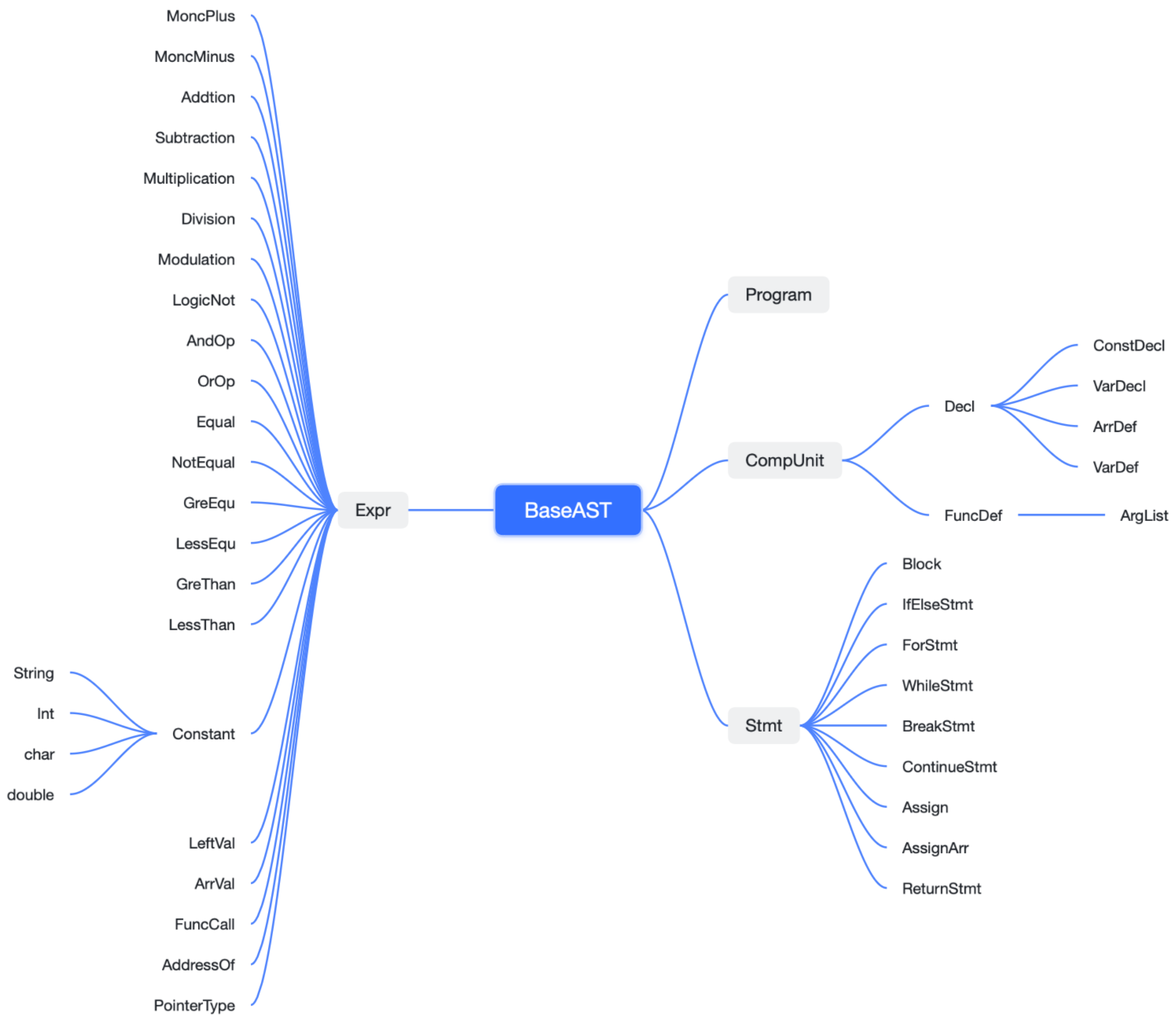
```
    : Decl                              { $$ = $1;}
    | Stmt                  { $$ = $1;}
    ;

SmooStmt
    : LVal ASSIGN Exp           { $$ = new AssignAST((LeftValAST*)$1, (ExprAST*)$3); }
    | Exp                   { $$ = $1; }
  | ArrValF ASSIGN Exp          { $$ = new AssignArrAST((ArrValAST*)$1, (ExprAST*)$3); }
    |                   { $$ = NULL; }
    | BREAK                             { $$ = new BreakStmtAST(); }
    | CONTINUE                              { $$ = new ContinueStmtAST(); }
    | RETURN RetState           { $$ = new ReturnStmtAST((ExprAST*)$2);}

Stmt
    : SmooStmt SEMI                 { $$ = $1; }
    | Block                 { $$ = $1; }
    | FOR LPAREN SmooStmt SEMI Exp SEMI SmooStmt RPAREN Block   { $$ = new
ForStmtAST((StmtAST*)$3, (ExprAST*)$5, (StmtAST*)$7, (BlockAST*)$9); }
    | IF LPAREN Exp RPAREN Block ElseState      { $$ = new IfElseStmtAST((ExprAST*)$3, (BlockAST*)$5,
(BlockAST*)$6); }
    | WHILE LPAREN Exp RPAREN Block             { $$ = new WhileStmtAST((ExprAST*)$3,
(BlockAST*)$5); }
    ;

LVal
    : IDENTIFIER                { $$ = new LeftValAST(*$1); }
    ;

ArrValF
  : IDENTIFIER ArrVal           { $$ = new ArrValAST(*$1, (Exprs*)$2); }

ArrVal
  : ArrVal LBRACKET Exp RBRACKET    { $$ = (Exprs*)$1; $$->push_back((ExprAST*)$3); }
  |                 { $$ = new Exprs(); }
  ;

ElseState
    : ELSE Block                        {$$ = $2; }
    |                               {$$ = NULL; }
    ;

RetState
    : Exp           {$$ = $1;}
    |                       {$$ = NULL; }
    ;

PrimaryExp
    : LPAREN Exp RPAREN       { $$ = $2; }
    | LVal              { $$ = $1; }
    | Constant            { $$ = $1; }
    ;

Constant
    : CONST_INT           { $$ = new Constant($1); }
    | CONST_CHAR           { $$ = new Constant($1); }
    | CONST_STR                      { $$ = new StringType(*$1); }
    ;

Exp
    : PrimaryExp           { $$ = $1; }
  | ArrValF            { $$ = (ArrValAST*)$1; }
    | ADD Exp %prec NOT        { $$ = new MoncPlus((ExprAST*)$2); }
    | SUB Exp %prec NOT        { $$ = new MoncMinus((ExprAST*)$2); }
    | NOT Exp            { $$ = new LogicNot((ExprAST*)$2); }
    | BAND LVal %prec NOT        { $$ = new AddressOf((LeftValAST*)$2); }

    | Exp ADD Exp         {$$ = new Addition((ExprAST*)$1, (ExprAST*)$3);}
    | Exp SUB Exp         {$$ = new Subtraction((ExprAST*)$1, (ExprAST*)$3);}
    | Exp MUL Exp         {$$ = new Multiplication((ExprAST*)$1, (ExprAST*)$3);}
    | Exp DIV Exp         {$$ = new Division((ExprAST*)$1, (ExprAST*)$3);}
    | Exp MOD Exp         {$$ = new Modulation((ExprAST*)$1, (ExprAST*)$3);}

    | Exp EQU Exp         {$$ = new Equal((ExprAST*)$1, (ExprAST*)$3);}
    | Exp NEQ Exp         {$$ = new NotEqual((ExprAST*)$1, (ExprAST*)$3);}
    | Exp LES Exp         {$$ = new LessThan((ExprAST*)$1, (ExprAST*)$3);}
    | Exp LEQ Exp         {$$ = new LessEqu((ExprAST*)$1, (ExprAST*)$3);}
    | Exp GRE Exp         {$$ = new GreThan((ExprAST*)$1, (ExprAST*)$3);}
    | Exp GEQ Exp         {$$ = new GreEqu((ExprAST*)$1, (ExprAST*)$3);}
    | Exp AND Exp         {$$ = new AndOp((ExprAST*)$1, (ExprAST*)$3);}
    | Exp OR  Exp        {$$ = new OrOp((ExprAST*)$1, (ExprAST*)$3);}

    | Exp BAND Exp
    | Exp BOR  Exp
    | Exp BXOR Exp

    | IDENTIFIER LPAREN ExpList RPAREN   {  $$ = new FuncCallAST(*$1, $3);   }
  ;

ExpList
    : _ExpList COMMA Exp                 {  $$ = $1; $$->push_back((ExprAST*)$3);   }
  | Exp                          {  $$ = new ExprListAST(); $$->push_back((ExprAST*)$1);   }
  |                    {  $$ = new ExprListAST();   }
  ;

_ExpList
    : _ExpList COMMA Exp               {  $$ = $1; $$->push_back((ExprAST*)$3);   }
  | Exp                          {  $$ = new ExprListAST(); $$->push_back((ExprAST*)$1);   }
  ;

ConstExp
  : CONST_INT            { $$ = new Constant($1); }
  ;
%%
```

2.2.4 抽象语法树（AST）

AST的结构和表示方式可以根据具体的语言和编译器实现而有所不同，但通常包含以下类型的节点：

1. 标识符（**Identifier**）节点：用于表示变量、函数名等标识符的节点。
2. 常量（**Const**）节点：用于表示常量值的节点，如整数、浮点数、字符串等。
3. 表达式（**Expression**）节点：用于表示各种表达式，如算术表达式、逻辑表达式、赋值表达式等。

4. 语句（**Statement**）节点：用于表示各种语句，如条件语句、循环语句、函数调用语句等。
5. 声明（**Declaration**）节点：用于表示变量声明、函数声明等语句。
6. 类型（**Type**）节点：用于表示变量或表达式的数据类型。
7. 控制流（**Control Flow**）节点：用于表示程序的控制流结构，如条件分支、循环等。



2.2.4.1 BaseAST类

BaseAST类是抽象语法树每个节点的纯虚类型，包括空的构造、析构函数和纯虚函数 IRGen 。

```
// 所有 AST 的基类
class BaseAST {
public:
 BaseAST() {}
   ~BaseAST() {}

   virtual llvm::Value* IRGen(IRGenerator& IRContext) = 0;
};
```

2.2.4.2 ProgramAST类

ProgramAST类是整个程序的根节点，本次使用中是为了将原来EBNF的文法转换成为BNF文法二产生的一个根节点。
而一个程序是由若干个函数声明（定义）和变量声明（定义组成），在我们的程序中，函数声明（定义）和变量声明（定义）具有共同的抽象类 CompUnitAST ，所以 ProgramAST 具有 std::vector<CompUnitAST*> 类型的成员变量。

```
class ProgramAST : public BaseAST {
public:
 CompUnits* compUnit_;

   ProgramAST(CompUnits* _compUnit_):compUnit_(_compUnit_){}
   ~ProgramAST(){};

   llvm::Value* IRGen(IRGenerator& IRContext);
};
```

2.2.4.3 DeclAST类

Decl类是纯虚类型，是普通变量和数组变量声明的父类。

1. 所有变量声明的抽象类父类

```
class DeclAST : public CompUnitAST {
public:
 DeclAST() {}
 ~DeclAST() {}

 virtual llvm::Value* IRGen(IRGenerator& IRContext) = 0;
};
```

2. 变量声明包括了变量定义列表和变量类型。

```
class VarDeclAST : public DeclAST {
public:
 VarDefAST* varDef_;
  VarType type_;

 VarDeclAST(std::string _typeName_, VarDefAST* _varDef_) :
  varDef_(_varDef_), type_(_typeName_) {}
 ~VarDeclAST() {}

 llvm::Value* IRGen(IRGenerator& IRContext);
};
```

3. 数组声明包括了基本元素类型、数组元素类型、数组名、数组索引列表。

```
class ArrDefAST : public BaseAST {
public:
 llvm::Type* elementType_;
 llvm::Type* arrayType_;
 std::string arrName_;
 VarType type_;
 Exprs* exprs_;



 ArrDefAST(std::string _typeName_, std::string _arrName_, Exprs* _exprs_) :
 type_(_typeName_), arrName_(_arrName_), exprs_(_exprs_) {}
 ~ArrDefAST() {}

 llvm::Value* IRGen(IRGenerator& IRContext);


};
```

2.2.4.4 StmtAST类

StmtAST类是无返回值的statement语句，是条件语句、循环语句、选择语句等子类的抽象父类。

```
class StmtAST: public BaseAST {
public:
  StmtAST() {}
  ~StmtAST() {}

    virtual llvm::Value* IRGen(IRGenerator& IRContext) = 0;
};
```

2.2.4.5 ExprAST类

ExprAST类是有返回值的expression语句，是常量表达式、变量表达式和操作符表达式等子类的抽象父类。

```
class ExprAST : public BaseAST {
public:
  ExprAST(void) {}
  ~ExprAST(void) {}

  virtual llvm::Value* IRGen(IRGenerator& IRContext) = 0;
};
```

2.2.4.6 BlockAST类

BlockAST类是基本语句块，而基本语句块包括stmt类语句和统计stmt数目。

```
class BlockAST : public BaseAST {
public:
    Stmts* stmts_;
  int varCnt_;

    BlockAST(Stmts* _stmts_): stmts_(_stmts_), varCnt_(0){}
    ~BlockAST(){}

  // void CreatePreDefinedVars(IRGenerator& IRContext);
    llvm::Value* IRGen(IRGenerator& IRContext);
};
```

2.2.4.7 VarType类

Varype是定义在BaseAST之外的类，用来定义和LLVM之间的数据类型链接。

```
class VarType {
public:
  PointerType* _BaseType_pointer;

    VarType(int) {type=Int;}
    VarType(char) {type=Char;}
  VarType(short) {type=Short;}
  VarType(double) {type=Double;}
  // VarType(ArrayType* __BaseType):_BaseType(__BaseType) {type=Arr;}
  VarType(PointerType* __BaseType):_BaseType_pointer(__BaseType) {type=Ptr;}
    VarType(std::string name);
    ~VarType(){}
    TypeID GetType() {return type;}
  llvm::Type* ToLLVMType(IRGenerator&);
private:
    TypeID type;
};

enum TypeID{
    Int,
    Char,
  Short,
  Double,
  Ptr
};
```

2.2.4.8 示例代码及对应AST结构介绍

示例代码:

```
int main(){
  int a = 10;
  int b[100];
  int c;
  b[10] = a;
  c = b[10] + a;
}
```

对应的AST结构:

```
- ProgramAST
  - ValDeclAST
    - BType        (int)
    - LeftValAST  (a)
    - Constant    (10)
  - ArrDeclAST
    - BType       (int)
  - ArrayType   (int[])
  - Constant    (100)
    - LeftValAST  (b)
  - ValDeclAST
    - TypeNode    (int)
  - AssignAST
  - LeftValAST  (a)
    - ArrValAST   (b[])
  - AssignAST     (+)
  - Addition
    - LeftValAST  (a)
    - ArrValAST   (b[])
  - LeftValAST   (c)
```

在这个简化的AST中，根节点是Program节点，表示整个程序。每个DeclarationStatement节点表示一个变量的声明语句，包含变量的类型、标识符和初始值（如果有）。BinaryExpressionNode节点表示两个标识符相加的表达式。

通过构建和遍历AST，编译器可以从源代码中提取出语法结构和语义信息，并进行后续的分析和转换操作。

2.2.5 语法分析器的测试和验证

Todo

1. *编写测试用例：编写一组全面而具有代表性的测试用例，覆盖语法分析器的不同语法规则和边界情况。测试用例应包括合法的输入代码和非法的输入代码，以验证语法分析器的正确性和错误处理能力。*
2. *语法规范验证：将测试用例与sysY语言的正式语法规范进行比对，确保语法分析器按照规范正确解析输入代码。检查是否有任何未处理的语法结构或语法错误。*
3. *边界情况测试：测试处理语法规范中定义的边界情况，例如最大允许的标识符长度、嵌套层数等。验证语法分析器在处理这些特殊情况时是否能够正确识别和处理。*
4. *错误处理测试：编写包含语法错误的测试用例，例如缺少分号、括号不匹配等。验证语法分析器能够准确地检测和报告这些错误，并给出恰当的错误信息。*
5. *性能测试：对语法分析器进行性能测试，包括处理大型源代码文件或具有复杂语法结构的代码。评估语法分析器在处理大规模代码时的效率和内存消耗情况。*
6. *与参考实现对比：如果有其他编译器或解析器实现sysY语言的参考实现，可以将测试用例同时运行在语法分析器和参考实现上，并对比其结果。这可以帮助发现潜在的问题或差异，并确保语法分析器的正确性。*
7. *随机测试：使用随机生成的代码片段作为输入，测试语法分析器的鲁棒性和稳定性。这可以帮助发现不常见的错误情况和边界条件。*
8. *自动化测试：建立自动化测试框架，包括编写自动化测试脚本和断言，以便能够快速运行和验证语法分析器的正确性。自动化测试可以提高测试的覆盖率和效率。*

2.3 语义分析设计

2.3.1 介绍

语义分析的主要目标是对源代码进行静态检查，发现和修复语义错误，并为后续阶段生成中间表示或目标代码提供准确的语义信息。

我们使用了LLVM的接口，手动将AST转化符合LLVM标准的IR代码

2.3.2 语义规则

2.3.2.1 类型检查

sysY语言是一种静态类型语言，因此在编译时会进行类型检查以确保操作的正确性。语义分析器将检查每个表达式和语句中的操作数和运算符之间的类型兼容性。对于算术运算符，要求操作数必须具有相同的类型或可以进行隐式类型转换。如果发现类型不匹配或不兼容的情况，语义分析器将生成相应的类型错误。

2.3.2.2 符号表和作用域规则

sysY语言使用块作用域规则，即在特定的代码块内声明的变量只在该块及其子块中可见。语义分析器将检查变量的作用域是否正确，并防止在非法的位置引用变量。对于变量的引用，语义分析器会检查变量是否在当前作用域内可见。

2.3.2.3 变量声明和使用规则

sysY语言要求在使用变量之前先进行声明。语义分析器将检查变量是否在使用之前已被声明，并且在同一作用域内不允许重复声明相同名称的变量。此外，语义分析器还会验证变量的类型和作用域等信息。

2.3.2.4 函数调用和参数匹配

在sysY语言中，函数调用要求参数的数量和类型与函数声明中的参数列表匹配。语义分析器将检查函数调用的参数数量和类型是否正确，并匹配对应的函数声明。如果存在参数不匹配的情况，语义分析器将生成相应的错误消息。

2.3.3 语义类型

2.3.3.1 Program类

Program::IRGen() 的作用是循环调用子结点的 IRGen() 。

```
llvm::Value* ProgramAST::IRGen(IRGenerator& IRContext) {

  for (auto compUnit : *(this->compUnit_)){
    if(compUnit){
      compUnit->IRGen(IRContext);
    }
  }

  return NULL;
}
```

2.3.3.2 Declare抽象类

2.3.3.2.1 VarDeclAST

该类负责变量声明的实现。在我们的实验环境下，变量定义分为全局变量和局部变量两种，结合LLVM的使用方式，需要对全局变量使用 new llvm::GlobalVariable() 进行创建，其中变量的信息则来自成员变量 varDef_ ，类型为 VarDef 。

```
llvm::Value* VarDeclAST::IRGen(IRGenerator& IRContext) {

  if (IRContext.GetCurFunc()) {
    // local variable

    auto IRBuilder = IRContext.IRBuilder;
    //创建变量
    auto AllocMem = IRBuilder->CreateAlloca(this->type_.ToLLVMType(IRContext), 0, this->varDef_-
>varName_);

    // llvm::Value* initVal = CastType(this->, IRContext)

    // initialize
    llvm::Value* value = this->varDef_->IRGen(IRContext);

    // store will always align to 4, even for char, which is because we need a type cast for 'value'
    IRBuilder->CreateStore(value, AllocMem);

    IRContext.CreateVar(this->type_, this->varDef_->varName_, AllocMem);
  }
  else {
    // global variable
    // initialize
    std::cout << "VarDeclAST -> global variable" << std::endl;

    llvm::Value* value = this->varDef_->IRGen(IRContext);

    // convert to const
    llvm::Constant* initializer = llvm::cast<llvm::Constant>(value);
    if (!initializer) {
      throw std::logic_error("The initializer is not const type: "+this->varDef_->varName_);
    }

    //Create a global variable
    auto AllocMem = new llvm::GlobalVariable(
      *(IRContext.Module),
      this->type_.ToLLVMType(IRContext),
      false,
      llvm::Function::ExternalLinkage,
      initializer,
      this->varDef_->varName_
    );

    IRContext.CreateVar(this->type_, this->varDef_->varName_, AllocMem);
  }

  return NULL;
}
```

2.3.3.2.2 VarDefAST

该类主要负责存储和创建变量的具体信息并转换成LLVM提供的变量类型，传给VarDeclAST进行变量声明。

```
llvm::Value* VarDefAST::IRGen(IRGenerator& IRContext) {
  if (this->initValue_) {
    //std::cout << "Have init" << std::endl;
    return this->initValue_->IRGen(IRContext);
  }
  else {
    auto IRBuilder = IRContext.IRBuilder;
    VarType* v = new VarType(this->varName_);
    switch(v->GetType()) {
    case Int: return IRBuilder->getInt32(0);
    case Char: return IRBuilder->getInt8(0);
    case Double:return llvm::ConstantFP::get(IRBuilder->getDoubleTy(), 0.0);
    }
  }
}
```

2.3.3.2.3 FuncDefAST

FuncDefAST 涉及到函数声明和函数定义两种，在该类的 IRFen() 函数中，有以下注意事项：

1. 函数名不能重复；
2. 区分是声明还是定义；
3. 确认返回值；
4. 参数列表的检查；
5. 构造函数基本块。

```
llvm::Value* FuncDefAST::IRGen(IRGenerator& IRContext) {
    //Get return type

    auto IRBuilder = IRContext.IRBuilder;
    llvm::Type* ReturnType = this->type_.ToLLVMType(IRContext);

    std::vector<llvm::Type*> ArgTypes;

  for (auto ArgType : *(this->_ArgList)) {
    llvm::Type* LLVMType = ArgType->type_.ToLLVMType(IRContext);
    if (!LLVMType) {
      throw std::logic_error("Defining a function " + this->funcName_ + " using unknown type(s).");
      return NULL;
    }
    ArgTypes.push_back(LLVMType);
  }

    //Get function type
    llvm::FunctionType* FuncType = llvm::FunctionType::get(ReturnType, ArgTypes, this->_ArgList-
>_VarArgLenth);

  if (this->block_) {
    // define function
    if (IRContext.IsFuncDefined(this->funcName_)) {
      throw std::logic_error("Function redeclared: "+this->funcName_);
    }

    llvm::Function* Func = IRContext.FindFunction(this->funcName_);
    if (Func) {
      IRContext.SetFuncDefined(this->funcName_);
    }
    else {
      Func = llvm::Function::Create(FuncType, llvm::Function::ExternalLinkage, this->funcName_,
IRContext.Module);
      IRContext.CreateFunc(FuncType, this->funcName_, Func, true);
    }

    int i = 0;
    for (auto ArgIter = Func->arg_begin(); ArgIter < Func->arg_end(); ArgIter++) {
      auto ArgInf = this->_ArgList->at(i);
      IRContext.RemainFutureVar(ArgInf->type_, ArgInf->_Name, ArgIter);
      i ++;
    }

    IRContext.SetCurFunc(Func);
    IRContext.ClearPreBrSignal();

    this->block_->IRGen(IRContext);

    IRContext.SetBasicBlock(NULL);
    IRContext.SetCurFunc(NULL);

  }
  else {
    // declare function
    if (IRContext.FindFunction(this->funcName_)) {
      // no need to declare again
      return NULL;
    }
    llvm::Function* Func = llvm::Function::Create(FuncType, llvm::Function::ExternalLinkage, this-
>funcName_, IRContext.Module);

    IRContext.CreateFunc(FuncType, this->funcName_, Func, false);
  }

    return NULL;
}
```

2.3.3.3 Stmt抽象类

Stmt抽象类被所有语句类继承：

```
Stmt ->    Expr SEMI |
      Assign SEMI |
      IfElseStmt |
      ForStmt |
      WhileStmt |
      BreakStmt |
      ContinueStmt |
      ReturnStmt |
      Block |
      SEMI
```

在本节中，我们将对 if else ， for ， while ， break ， continue ， return 语句以及语句块进行语义分析和代码生成的介绍。

2.3.3.3.1 IfElseStmtAST

```
IfStmt ->   IF LPAREN Exp RPAREN Block ElseState
ElseState ->   ELSE Block |
```

if 语句的代码生成，基本思路如下：

If中包含三个主要的部分：条件，if的语句块，else的语句块。我们以Basic Block来组织代码生成的过程。先在原来的block里解码判断条件，形成条件分支语句。之后，分别解码if和else对应的语句块。由于if语句需要一个出口，且最后离开这部分代码的时候，当前模块所在的basic block需要处于一个新的block，所以还需额外创建一个 Outblock ，作为这段代码的出口，if和else的语句块都要指向它。

```
llvm::Value* IfElseStmtAST::IRGen(IRGenerator& IRContext) {
  std::cout << "IfElseStmtAST" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;

  auto CondExpr = this->cond_->IRGen(IRContext);
  llvm::BasicBlock* CondBlock = IRBuilder->GetInsertBlock();

  IRContext.ClearPreBrSignal();
  llvm::BasicBlock* IfBlock = (llvm::BasicBlock*)this->ifBlock_->IRGen(IRContext);
  llvm::BasicBlock* IfOutBlock = IRBuilder->GetInsertBlock();
  llvm::BasicBlock* ElseBlock, *ElseOutBlock;
  if (this->elseBlock_) {
    IRContext.ClearPreBrSignal();
    ElseBlock = (llvm::BasicBlock*)this->elseBlock_->IRGen(IRContext);
    ElseOutBlock = IRBuilder->GetInsertBlock();
  }

  // set exit
  llvm::Function* Func = IRContext.GetCurFunc();
  llvm::BasicBlock* OutBlock = llvm::BasicBlock::Create(*(IRContext.Context), "BBExit", Func);
  IRBuilder->SetInsertPoint(IfOutBlock);
  IRBuilder->CreateBr(OutBlock);
  if (this->elseBlock_) {
    IRBuilder->SetInsertPoint(ElseOutBlock);
    IRBuilder->CreateBr(OutBlock);
  }

  // set conditional branch
  IRBuilder->SetInsertPoint(CondBlock);
  IRBuilder->CreateCondBr(CondExpr, IfBlock, this->elseBlock_?ElseBlock:OutBlock);

  IRBuilder->SetInsertPoint(OutBlock);

  return NULL;
}
```

2.3.3.3.2 ForStmtAST

for 语句的结构如下：

```
ForStmt ->    FOR LPAREN Stmt SEMI Exp SEMI Stmt RPAREN Block
```

for 语句的代码生成实现如下：

```
llvm::Value* ForStmtAST::IRGen(IRGenerator& IRContext) {
  std::cout << "ForStmtAST" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;
  llvm::Function* Func = IRContext.GetCurFunc();
  // init generate
  if (this->initStmt_)
    this->initStmt_->IRGen(IRContext);
  llvm::BasicBlock* cmpBlock = llvm::BasicBlock::Create(*(IRContext.Context), "ForCmp", Func);
  llvm::BasicBlock* iterBlock = llvm::BasicBlock::Create(*(IRContext.Context), "ForIter", Func);
  llvm::BasicBlock* exitBlock = llvm::BasicBlock::Create(*(IRContext.Context), "ForExit", Func);
  IRBuilder->CreateBr(cmpBlock);
  // enter the Loop
  IRContext.EnterLoop(cmpBlock, iterBlock, exitBlock);

  // condition generate
  IRBuilder->SetInsertPoint(cmpBlock);
  auto cmpRes = (this->condExpr_)?this->condExpr_->IRGen(IRContext):IRBuilder->getInt1(true);

  // body generate
  IRContext.ClearPreBrSignal();
  llvm::BasicBlock* bodyBlock = (llvm::BasicBlock*)this->forBody_->IRGen(IRContext);
  llvm::BasicBlock* bodyOutBlock = IRBuilder->GetInsertBlock();

  // iteration generate
  IRBuilder->CreateBr(iterBlock);
  IRBuilder->SetInsertPoint(iterBlock);
  if (this->iterStmt_) this->iterStmt_->IRGen(IRContext);
  IRBuilder->CreateBr(cmpBlock);

  // set conditional branch
  IRBuilder->SetInsertPoint(cmpBlock);
  IRBuilder->CreateCondBr(cmpRes, bodyBlock, exitBlock);

  // leave the Loop
  IRContext.LeaveCurrentLoop();
  IRBuilder->SetInsertPoint(exitBlock);

  return NULL;
}
```

2.3.3.3.3 WhileStmtAST

while 语句的结构如下：

```
WhileStmt ->  WHILE LPAREN Exp RPAREN Block
```

while 语句的代码生成如下：

```
llvm::Value* WhileStmtAST::IRGen(IRGenerator& IRContext) {
  std::cout << "WhileStmtAST" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;
  llvm::Function* Func = IRContext.GetCurFunc();

  // init
  llvm::BasicBlock* cmpBlock = llvm::BasicBlock::Create(*(IRContext.Context), "WhileCmp", Func);
  llvm::BasicBlock* exitBlock = llvm::BasicBlock::Create(*(IRContext.Context), "WhileExit", Func);
  IRBuilder->CreateBr(cmpBlock);
  // enter the Loop
  IRContext.EnterLoop(cmpBlock, NULL, exitBlock);

  // condition generate
  IRBuilder->SetInsertPoint(cmpBlock);
  auto cmpRes = (this->condExpr_)?this->condExpr_->IRGen(IRContext):IRBuilder->getInt1(true);

  // body generate
  IRContext.ClearPreBrSignal();
  llvm::BasicBlock* bodyBlock = (llvm::BasicBlock*)this->whileBody_->IRGen(IRContext);
  llvm::BasicBlock* bodyOutBlock = IRBuilder->GetInsertBlock();
  IRBuilder->CreateBr(cmpBlock);

  // set conditional branch
  IRBuilder->SetInsertPoint(cmpBlock);
  IRBuilder->CreateCondBr(cmpRes, bodyBlock, exitBlock);

  // leave the Loop
  IRContext.LeaveCurrentLoop();
  IRBuilder->SetInsertPoint(exitBlock);

  return NULL;
}
```

2.3.3.3.4 BreakStmtAST

break 语句的代码生成如下：

```
llvm::Value* BreakStmtAST::IRGen(IRGenerator& IRContext) {
  std::cout << "BreakStmtAST" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;
  llvm::BasicBlock* targetBlock = IRContext.BreakCurrentLoop();
  IRBuilder->CreateBr(targetBlock);

  return NULL;
}
```

2.3.3.3.5 ContinueStmtAST

continue 语句的代码生成如下：

```
llvm::Value* ContinueStmtAST::IRGen(IRGenerator& IRContext) {
  std::cout << "ContinueStmtAST" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;
  llvm::BasicBlock* targetBlock = IRContext.ContinueCurrentLoop();
  IRBuilder->CreateBr(targetBlock);

  return NULL;
}
```

2.3.3.3.6 ReturnStmtAST

return 语句的代码生成如下：

```
llvm::Value* ReturnStmtAST::IRGen(IRGenerator& IRContext) {
    std::cout << "ReturnAST" << std::endl;
    auto IRBuilder = IRContext.IRBuilder;
  if (this->RetVal_)
      IRBuilder->CreateRet(this->RetVal_->IRGen(IRContext));
  else
    IRBuilder->CreateRetVoid();
    return NULL;
}
```

2.3.3.4 Expr抽象类

本次实验中为了方便实验设计，对C语言中的表达式进行了更加严格的限制（参考SysY语言进行设计），具体要求如下：

- 赋值时，表达式左值必须是变量，例如: `a,b,c,d[1][2]`；
- 赋值时，表达式的左值需要提供有一个存储空间；
- 赋值时，表达式右值必须具有返回值，本次实验中具体为一般的逻辑运算、大小比较、算术运算；
- 非赋值时，可以是任意支持的表达式。

而在LLVM中，所有创建的变量都是通过提供指向这一变量存储空间的指针来实现的，当要获取变量的值时通过 `CreateLoad()` 进行获取，当要改变变量的值时通过 `CreateStore()` 来实现。因此在我们实现的编译器中，对上述提到的合法左值提供一个 `IRGenPtr()` 函数以获取指向存储空间的指针，对所有表达式提供 `IRGen()` 函数来获取其存储空间内的值。

2.3.3.4.1 支持字面量

我们编译器支持以下字面量：

- INTEGER
- CHAR
- DOUBLE
- STRING

2.3.3.4.2 右值支持类

如上文所说，一般的右值支持类只需要返回结果而不需要提供存储地址，因此只需要实现 IRGen() 函数即可。

1. 算术运算

本次实验中实现了基本的算术运算，包括加、减、乘、除、取模、单目加、单目减。
在 IRGen() 中需要注意类型转换，该部分的程序结构如下：

```
llvm::Value* Addition::IRGen(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  llvm::Value* LHS = this->LHS_->IRGen(IRContext);
  llvm::Value* RHS = this->RHS_->IRGen(IRContext);
  if (LHS->getType()->isIntegerTy())
    return IRBuilder->CreateAdd(LHS, RHS);
  else
    return IRBuilder->CreateFAdd(LHS, RHS);
}
```

2. 逻辑运算

本次实验中实现了基本的逻辑运算，包括取反操作、与操作、或操作。在 IRGen() ，该部分的程序结构如下：

```
llvm::Value* AndOp::IRGen(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  llvm::Value* LHS = ToBoolType(this->LHS_->IRGen(IRContext), IRContext);
  llvm::Value* RHS = ToBoolType(this->RHS_->IRGen(IRContext), IRContext);
  return IRBuilder->CreateLogicalAnd(LHS, RHS);
}
```

3. 比较运算

本次实验中实现了基本的比较运算，包括等于、不等于、大于、小于、大于等于、小于等于。
在 IRGen() ，该部分的程序结构如下：

```
llvm::Value* Equal::IRGen(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  llvm::Value* LHS = this->LHS_->IRGen(IRContext);
  llvm::Value* RHS = this->RHS_->IRGen(IRContext);
  return IRBuilder->CreateICmpEQ(LHS, RHS);
}
```

4. 函数调用

本实验中的函数调用支持的参数包括变量传参和数组传指针。

```
llvm::Value* FuncCallAST::IRGen(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  llvm::Function* Func = IRContext.FindFunction(this->_FuncName);



  if (Func == NULL) {
    throw std::domain_error(this->_FuncName + " is not a defined function.");
    return NULL;
  }

  if (Func->isVarArg() && this->_ArgList->size() < Func->arg_size() ||
    !Func->isVarArg() && this->_ArgList->size() != Func->arg_size()) {
    throw std::invalid_argument("Args doesn't match when calling function " + this->_FuncName + ". Expected "
+ std::to_string(Func->arg_size()) + ", got " + std::to_string(this->_ArgList->size()));
    return NULL;
  }

  std::vector<llvm::Value*> ArgList;
  size_t Index = 0;
  for (auto ArgIter = Func->arg_begin(); ArgIter < Func->arg_end(); ArgIter++, Index++) {
    llvm::Value* Arg = this->_ArgList->at(Index)->IRGen(IRContext);
    Arg = TypeCasting(Arg, ArgIter->getType(), IRContext);
    if (Arg == NULL) {
      throw std::invalid_argument(std::to_string(Index) + "-th arg type doesn't match when calling function " +
this->_FuncName + ".");
      return NULL;
    }
    ArgList.push_back(Arg);
  }
  if (Func->isVarArg())
   for (; Index < this->_ArgList->size(); Index++) {
    llvm::Value* Arg = this->_ArgList->at(Index)->IRGen(IRContext);
    if (Arg->getType()->isIntegerTy())
      Arg = TypeUpgrading(Arg, IRBuilder->getInt32Ty(), IRContext);
    else if (Arg->getType()->isFloatingPointTy())
      Arg = TypeUpgrading(Arg, IRBuilder->getDoubleTy(), IRContext);
    ArgList.push_back(Arg);
   }

  return IRBuilder->CreateCall(Func, ArgList);
}
```

2.3.3.4.3 左值支持类

左值在满足其作为右值的基础上，需要提供一个指向存储空间的指针，因此需要设计 IRGenPtr() 函数以获取指向存储空间的指针。

1. LeftValAST类

LeftValAST 类表示一般的变量，当它为左值时，由于LLVM里是通过指针来实现，因此结果存储需要提供一个指向存储地址的指针，因此为 leftValAST 类提供一个 IRGenPtr() 函数来获取指针。

```
llvm::Value* LeftValAST::IRGenPtr(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  llvm::Value* VarPtr = IRContext.FindVar(this->name_);
  return VarPtr;
}
```

当它为右值时（即需要在运算中提供值），其 IRGen() 函数在获取指针的基础上使用 CreateLoad() 函数将值提取到 LLVM::Value* 中。

```
llvm::Value* LeftValAST::IRGen(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  llvm::Value* VarPtr = IRContext.FindVar(this->name_);
  if (IRContext.IsPtrVar(this->name_)) {
    return VarPtr;
  }
  else {
    llvm::Type* type = VarPtr->getType()->getNonOpaquePointerElementType();
    llvm::Value* val = IRBuilder->CreateLoad(type,VarPtr);
    return val;
  }
}
```

2. ArrValAST类

ArrValAST 类表示数组类变量。数组的表示也是本次实验中的一个难点，通过查询LLVM的相关文档，LLVM构造数组的方式是以基础类作为元素封装成更高维，根据维度决定封装层数；而重点是数组元素的获取，LLVM提供了两种方式，一种是通过一个指向原数组的指针然后提供所有索引直接获取，还有一种是指针层层深入，降低维度，最终指向具体元素。在具体的实现过程中，我们选择了后者进行实现，因为我们在实验中发现前者虽然能够产生合理的IR，但是产生的IR在生成目标代码。

以下是该类的 IRGen() 。

```
llvm::Value* ArrValAST::IRGenPtr(IRGenerator& IRContext) {
  std::cout << "ArrValPtr" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;

  //搜索数组的指针
  llvm::Value* arrayPtr = IRContext.FindVar(this->name_);
  arrayPtr->print(llvm::outs());

  //this->exprs_ index索引

  std::vector<llvm::Value*> indices;

  //生成每个维度的索引

  for(auto expr : *(this->exprs_)){
    indices.push_back(expr->IRGen(IRContext));

  }

  llvm::Value* v1, *v2;

  for(auto indice: indices){
    if(arrayPtr->getType()->getNonOpaquePointerElementType()->isArrayTy()){
      v1 = IRBuilder->CreatePointerCast(arrayPtr, arrayPtr->getType()->getNonOpaquePointerElementType()-
>getArrayElementType()->getPointerTo());
    }
    else if(arrayPtr->getType()->isPointerTy()){
      v1 = IRBuilder->CreateLoad(arrayPtr->getType()->getNonOpaquePointerElementType(), arrayPtr);
    }
    else{
      throw std::logic_error("The sunsciption operation received neither array type nor pointer type");
    }
    v2 = IRBuilder->CreateGEP(v1->getType()->getNonOpaquePointerElementType(), v1, indice);
  }
  return v2;
}
```

以下是该类的 IRGen()， 该函数实际上是在找到指向地址的基础上调用 CreateLoad() 进行实现。

```
llvm::Value* ArrValAST::IRGen(IRGenerator& IRContext) {
  std::cout << "ArrVal" << std::endl;

  auto IRBuilder = IRContext.IRBuilder;

  //搜索数组的指针
  llvm::Value* arrayPtr = IRContext.FindVar(this->name_);
  // auto* arrayPtr = IRContext.Module->getGlobalVariable(this->name_);
  std::cout << "Array Ptr " << arrayPtr << std::endl;

  //this->exprs_ index索引

  std::vector<llvm::Value*> indices;
  //生成每个维度的索引

  for(auto expr : *(this->exprs_)){
    indices.push_back(expr->IRGen(IRContext));

  }

  llvm::Value* v1, *v2;

  for(auto indice: indices){
    if(arrayPtr->getType()->getNonOpaquePointerElementType()->isArrayTy()){
      v1 = IRBuilder->CreatePointerCast(arrayPtr, arrayPtr->getType()->getNonOpaquePointerElementType()-
>getArrayElementType()->getPointerTo());
    }
    else if(arrayPtr->getType()->isPointerTy()){
      v1 = IRBuilder->CreateLoad(arrayPtr->getType()->getNonOpaquePointerElementType(), arrayPtr);
    }
    else{
      throw std::logic_error("The sunsciption operation received neither array type nor pointer type");
    }
    // v1 = IRBuilder->CreatePointerCast(arrayPtr, arrayPtr->getType()->getNonOpaquePointerElementType()-
>getArrayElementType()->getPointerTo());

    v2 = IRBuilder->CreateGEP(v1->getType()->getNonOpaquePointerElementType(), v1, indice);

  }

  return IRBuilder->CreateLoad(v1->getType()->getNonOpaquePointerElementType(), v2);

}
```

2.3.3.5 VarType抽象类

本次实验中，我们将 VarType() 类定义独立在 BaseAST() 之外，由于函数类型在lexer和parser中是以字符串的形式传递的，所以需要根据字符串构建出相应的数据类型，同时匹配对应的 LLVM::Type 。
以下是先根据字符串匹配对应的枚举值。

```
VarType::VarType(std::string name) {
  if (name == "int") type = Int;
  else if (name == "char") type = Char;
  else if (name == "short") type = Short;
  else if (name == "double") type = Double;
}
```

以下是根据枚举值生成对应的LLVM::Type。

```
llvm::Type* VarType::ToLLVMType(IRGenerator& IRContext) {
  auto IRBuilder = IRContext.IRBuilder;
  switch(this->type) {
    case Int: return IRBuilder->getInt32Ty();
    case Char: return IRBuilder->getInt8Ty();
    case Short: return IRBuilder->getInt16Ty();
    case Double: return IRBuilder->getDoubleTy();
    case Ptr: return this->_BaseType_pointer->ToLLVMType(IRContext);
  }
}
```

2.3.4 语义错误处理

说明语义分析器如何检测和报告语义错误，包括类型错误、未声明的变量、重复声明等。解释语义错误的分类和处理策略，如报错、警告或修复。

2.4 运行环境设计

2.4.1 IR上下文环境结构体

```
    llvm::LLVMContext* Context;
    llvm::IRBuilder<>* IRBuilder;
    llvm::Module* Module;

    llvm::Function* curFunc_;
    BlockAST* curBasicBlock_;
    bool bbCreatePreBrSignal_;
    std::vector<IRVarAttr*> varList_;
    std::vector<IRFuncAttr*> funcList_;
    std::vector<IRLoopAttr*> loopLevel_;
    std::vector<IRVarAttr*> varListForFuture_;
```

2.4.2 创建IR模块

```
    IRGenerator(){
        Context = new llvm::LLVMContext;
        IRBuilder = new llvm::IRBuilder<>(*Context);
        Module = new llvm::Module("main", *Context);

        bbCreatePreBrSignal_ = true;
        curBasicBlock_ = NULL;
        curFunc_ = NULL;
    }
```

2.4.3 符号表

```

void IRGenerator::CreateVar(VarType type, std::string name, llvm::Value* value, bool isPtr){
    // first judge name
    int conflictCnt;
    int varCnt = this->varList_.size();
    if (this->curBasicBlock_) conflictCnt = this->curBasicBlock_->varCnt_;
    else conflictCnt = varCnt;

    // std::cout << "CreateVar " << conflictCnt << " " << varCnt << std::endl;
    for (int i = 1; i <= conflictCnt; i++) {
        if (this->varList_[varCnt-i]->name_ == name) {
            // already has the same name
            throw std::logic_error("Already Create a Variable with Name: "+name);
        }
    }

    this->varList_.push_back(new IRVarAttr(type, name, value, isPtr));
    if (this->curBasicBlock_) this->curBasicBlock_->varCnt_ += 1;
}

void IRGenerator::DiscardVar() {
    if (this->curBasicBlock_)
        for (int i = 0; i < this->curBasicBlock_->varCnt_; i++) {
            auto var = this->varList_[this->varList_.size()-1];
            this->varList_.pop_back();
            delete var;
        }
}

llvm::Value* IRGenerator::FindVar(std::string name){
  if(this->varList_.size() == 0){
    return NULL;
  }
  for(auto symbol = this->varList_.end() - 1; symbol >= this->varList_.begin(); symbol--){
        // std::cout << "FindVar " << (*symbol)->name_ << " " << name << std::endl;
    if((*symbol)->name_ == name){
      return (*symbol)->value_;
    }
  }
  return NULL;
}

bool IRGenerator::IsPtrVar(std::string name) {
    if(this->varList_.size() == 0){
    return false;
  }
  for(auto symbol = this->varList_.end() - 1; symbol >= this->varList_.begin(); symbol--){
    if((*symbol)->name_ == name){
      return (*symbol)->isPtr_;
    }
  }

  return false;
}
```

2.4.4 当前状态设置

```
void IRGenerator::SetCurFunc(llvm::Function* curFunc) {
    this->curFunc_ = curFunc;
}

llvm::Function* IRGenerator::GetCurFunc() {
    return this->curFunc_;
}

void IRGenerator::SetPreBrSignal() {
    this->bbCreatePreBrSignal_ = true;
}

bool IRGenerator::ClearPreBrSignal() {
    bool bbCreatePreBrSignal = this->bbCreatePreBrSignal_;
    this->bbCreatePreBrSignal_ = false;
    return bbCreatePreBrSignal;
}

bool IRGenerator::GetPreBrSignal() {
    return this->bbCreatePreBrSignal_;
}

BlockAST* IRGenerator::GetBasicBlock() {
    return this->curBasicBlock_;
}
```

```
void IRGenerator::SetBasicBlock(BlockAST* newBasicBlock){
    this->curBasicBlock_ = newBasicBlock;
}
```

2.4.5 循环结构

```
void IRGenerator::EnterLoop(llvm::BasicBlock* condBlock, llvm::BasicBlock* iterBlock, llvm::BasicBlock*
exitBlock) {
    this->loopLevel_.push_back(new IRLoopAttr(condBlock, iterBlock, exitBlock));
}

void IRGenerator::LeaveCurrentLoop() {
    this->loopLevel_.pop_back();
}

llvm::BasicBlock* IRGenerator::BreakCurrentLoop() {
    auto currentLoop = this->loopLevel_[this->loopLevel_.size()-1];
    return currentLoop->exitBlock_;
}

llvm::BasicBlock* IRGenerator::ContinueCurrentLoop() {
    auto currentLoop = this->loopLevel_[this->loopLevel_.size()-1];
    return currentLoop->iterBlock_?currentLoop->iterBlock_:currentLoop->condBlock_;
}
```

2.4.6 函数管理

```
void IRGenerator::CreateFunc(llvm::FunctionType* type, std::string name, llvm::Function* func, bool
isDefined){
    if(this->FindFunction(name)){
        return;
    }
    this->funcList_.push_back(new IRFuncAttr(type, name, func, isDefined));
}
```

```
void IRGenerator::DiscardFunc(int cnt) {
    for (int i = 0; i < cnt; i++) {
        auto func = this->funcList_[this->funcList_.size()-1];
    this->funcList_.pop_back();
        delete func;
    }
}

llvm::Function* IRGenerator::FindFunction(std::string Name) {
  if (this->funcList_.size() == 0) return NULL;
    for (auto iter = this->funcList_.end()-1; iter >= this->funcList_.begin(); iter--) {
        if ((*iter)->getName() == Name){
            return (*iter)->getFunc();
        }
    }
  return NULL;
}

bool IRGenerator::IsFuncDefined(std::string Name) {
  if (this->funcList_.size() == 0) return false;
    for (auto iter = this->funcList_.end()-1; iter >= this->funcList_.begin(); iter--) {
        if ((*iter)->getName() == Name){
            if ((*iter)->getDefined()) return true;
            else return false;
        }
    }
  return false;
}

bool IRGenerator::SetFuncDefined(std::string Name) {
    if (this->funcList_.size() == 0) return false;
    for (auto iter = this->funcList_.end()-1; iter >= this->funcList_.begin(); iter--) {
        if ((*iter)->getName() == Name){
            (*iter)->setDefined();
            return true;
        }
    }
  return false;
}
```

2.5 代码生成设计

首先实例化 llvm::sys::getDefaultTargetTriple ， 该对象包含了目标机器的许多参数。然后我们调用 llvm::TargetMachine 的接口即可把LLVM中间代码编译成目标机器的汇编代码。具体请参考LLVM官方文档。

```
void IRGenerator::GenObjectCode(std::string outputfile) {
  //获取描述编译器的目标平台、操作系统和环境等信息
  auto TargetTriple = llvm::sys::getDefaultTargetTriple();
  //根据llvm文档提供的初始化
    llvm::InitializeAllTargetInfos();
    llvm::InitializeAllTargets();
    llvm::InitializeAllTargetMCs();
    llvm::InitializeAllAsmParsers();
    llvm::InitializeAllAsmPrinters();

  //根据TargetTriple查找目标平台
    std::string Error;
    auto Target = llvm::TargetRegistry::lookupTarget(TargetTriple, Error);

    llvm::TargetOptions opt;
    auto RM = llvm::Optional<llvm::Reloc::Model>();
    auto TargetMachine = Target->createTargetMachine(TargetTriple, "", "", opt, RM);
  //设置数据布局
    Module->setDataLayout(TargetMachine->createDataLayout());
    Module->setTargetTriple(TargetTriple);

  //将数据写入文件
  std::error_code errCode;
    llvm::raw_fd_ostream OPFile(outputfile, errCode, llvm::sys::fs::OF_None);
    llvm::legacy::PassManager PM;
    TargetMachine->addPassesToEmitFile(PM, OPFile, nullptr, llvm::CGFT_ObjectFile);
    PM.run(*Module);
    OPFile.flush();
  }
```

2.6 符号表设计

LLVM本身不支持不同作用阈内的变量重名（它会把重名变量自动添加序号），因此在本次实验中，我们通过构造一个栈来实现符号表，以保证变量作用阈的合理性。

以下是符号表的结构，首先需要定义栈的元素类型，需要说明的是，我们是通过变量名来搜索，得到指向元素的指针。

```
class IRGenerator {
public:
  ...
  std::vector<IRVarAttr*> varList_;
  ...
}
class IRVarAttr {
public:
    VarType type_;
    std::string name_;
    bool isPtr_;

  //llvm::Function* CurFunc;
    llvm::Value* value_;

    IRVarAttr(VarType type, std::string name, llvm::Value* value, bool _isPtr_=false):
        type_(type), name_(name), value_(value), isPtr_(_isPtr_){}
};
```

以下是符号表的相关操作：

1. 创建变量存入符号表

```
void IRGenerator::CreateVar(VarType type, std::string name, llvm::Value* value, bool isPtr){
    // first judge name
    int conflictCnt;
    int varCnt = this->varList_.size();
    if (this->curBasicBlock_) conflictCnt = this->curBasicBlock_->varCnt_;
    else conflictCnt = varCnt;

    // std::cout << "CreateVar " << conflictCnt << " " << varCnt << std::endl;
    for (int i = 1; i <= conflictCnt; i++) {
        if (this->varList_[varCnt-i]->name_ == name) {
            // already has the same name
            throw std::logic_error("Already Create a Variable with Name: "+name);
        }
    }

    this->varList_.push_back(new IRVarAttr(type, name, value, isPtr));
    if (this->curBasicBlock_) this->curBasicBlock_->varCnt_ += 1;
}
```

2. 查找符号

```
llvm::Value* IRGenerator::FindVar(std::string name){
  if(this->varList_.size() == 0){
    return NULL;
  }
  for(auto symbol = this->varList_.end() - 1; symbol >= this->varList_.begin(); symbol--){
      // std::cout << "FindVar " << (*symbol)->name_ << " " << name << std::endl;
    if((*symbol)->name_ == name){
      return (*symbol)->value_;
    }
  }
  return NULL;
}
```

3. 删除符号

```
void IRGenerator::DiscardVar() {
    if (this->curBasicBlock_)
        for (int i = 0; i < this->curBasicBlock_->varCnt_; i++) {
            auto var = this->varList_[this->varList_.size()-1];
            this->varList_.pop_back();
            delete var;
        }
}
```

4. 指针查找

```
bool IRGenerator::IsPtrVar(std::string name) {
    if(this->varList_.size() == 0){
    return false;
  }
  for(auto symbol = this->varList_.end() - 1; symbol >= this->varList_.begin(); symbol--){
    if((*symbol)->name_ == name){
      return (*symbol)->isPtr_;
    }
  }

  return false;
}
```

三、运行结果

请附上你的源语言在你的编译器下的产生二进制或其他东西的过程，以及在第二节功能测试中的结果。

3.1 QuickSort

测试代码:

```
int printk(int ptr, ...);
int scank(char ptr, ...);

int quicksort(int ptr a, int left, int right){
  //printk("QuickSort\n");
  int i = left;
  int j = right;
  int temp = i + j;
  temp = temp/2;
  //printk(" %d %d %d @@@", temp, left, right);
  int mid;
  mid = a[temp];
  //printk(" %d# ", mid);
  while (i < j){
    while (a[i] < mid) {
        i = i + 1;
      //printk("What i%d?\n",i);
        }
    while (mid < a[j]){
        j = j - 1;
      //printk("What j%d?\n",j);
        }
    if (i <= j){
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
      i = i + 1;
        j = j - 1;
    }
    //printk("\n");
  }

  if (left < j) {
      quicksort(a, left, j);
    }
  if (i < right) {
        quicksort(a, i, right);
    }

    return 0;
}

int main(){
  int n;
    int temp;
  int a[10000];
  scank("%d", &n);
    int i;

  for (i = 0; i < n; i = i + 1){
        scank("%d", &temp);
        a[i] = temp;
    }

  quicksort(a, 0, n - 1);
  for (i = 0; i < n; i=i+1){
    printk("%d\n", a[i]);
    }

  return 0;
}
```

IR代码：

```
; ModuleID = 'main'
source_filename = "main"

@0 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
```

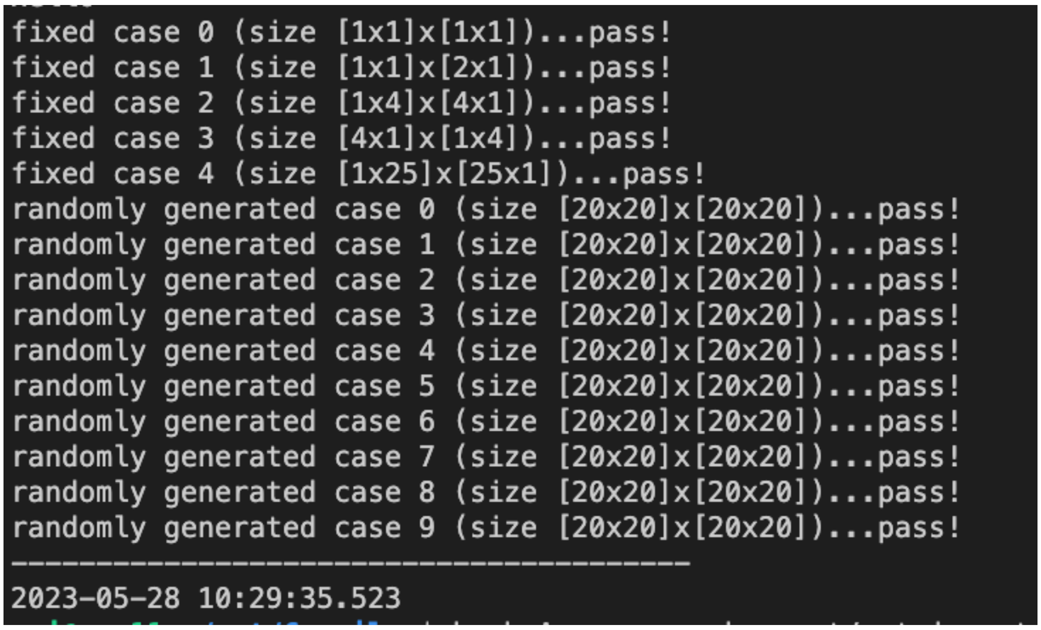
```
declare i32 @printk(i32*, ...)

declare i32 @scank(i8*, ...)

define i32 @quicksort(i32* %0, i32 %1, i32 %2) {
BBEntry:
  %right = alloca i32, align 4
  store i32 %2, i32* %right, align 4
  %left = alloca i32, align 4
  store i32 %1, i32* %left, align 4
  %a = alloca i32*, align 8
  store i32* %0, i32** %a, align 8
  %i = alloca i32, align 4
  %3 = load i32, i32* %left, align 4
  store i32 %3, i32* %i, align 4
  %j = alloca i32, align 4
  %4 = load i32, i32* %right, align 4
  store i32 %4, i32* %j, align 4
  %temp = alloca i32, align 4
  %5 = load i32, i32* %i, align 4
  %6 = load i32, i32* %j, align 4
  %7 = add i32 %5, %6
  store i32 %7, i32* %temp, align 4
  %8 = load i32, i32* %temp, align 4
  %9 = sdiv i32 %8, 2
  store i32 %9, i32* %temp, align 4
  %mid = alloca i32, align 4
  store i8 0, i32* %mid, align 1
  %10 = load i32, i32* %temp, align 4
  %11 = load i32*, i32** %a, align 8
  %12 = getelementptr i32, i32* %11, i32 %10
  %13 = load i32, i32* %12, align 4
  store i32 %13, i32* %mid, align 4
  br label %WhileCmp

WhileCmp:                                         ; preds = %BBExit, %BBEntry
  %14 = load i32, i32* %i, align 4
  %15 = load i32, i32* %j, align 4
  %16 = icmp slt i32 %14, %15
  br i1 %16, label %BBEntry1, label %WhileExit

WhileExit:                                ; preds = %WhileCmp
  %17 = load i32, i32* %left, align 4
  %18 = load i32, i32* %j, align 4
  %19 = icmp slt i32 %17, %18
  br i1 %19, label %BBEntry10, label %BBExit11

BBEntry1:                                 ; preds = %WhileCmp
  br label %WhileCmp2

WhileCmp2:                                ; preds = %BBEntry4, %BBEntry1
  %20 = load i32, i32* %i, align 4
  %21 = load i32*, i32** %a, align 8
  %22 = getelementptr i32, i32* %21, i32 %20
  %23 = load i32, i32* %22, align 4
  %24 = load i32, i32* %mid, align 4
  %25 = icmp slt i32 %23, %24
  br i1 %25, label %BBEntry4, label %WhileExit3

WhileExit3:                               ; preds = %WhileCmp2
  br label %WhileCmp5

BBEntry4:                                 ; preds = %WhileCmp2
  %26 = load i32, i32* %i, align 4
  %27 = add i32 %26, 1
  store i32 %27, i32* %i, align 4
  br label %WhileCmp2

WhileCmp5:                                ; preds = %BBEntry7, %WhileExit3
  %28 = load i32, i32* %mid, align 4
  %29 = load i32, i32* %j, align 4
  %30 = load i32*, i32** %a, align 8
  %31 = getelementptr i32, i32* %30, i32 %29
  %32 = load i32, i32* %31, align 4
  %33 = icmp slt i32 %28, %32
  br i1 %33, label %BBEntry7, label %WhileExit6

WhileExit6:                               ; preds = %WhileCmp5
```

```
  %34 = load i32, i32* %i, align 4
  %35 = load i32, i32* %j, align 4
  %36 = icmp sle i32 %34, %35
  br i1 %36, label %BBEntry8, label %BBExit

BBEntry7:                                   ; preds = %WhileCmp5
  %37 = load i32, i32* %j, align 4
  %38 = sub i32 %37, 1
  store i32 %38, i32* %j, align 4
  br label %WhileCmp5

BBEntry8:                                   ; preds = %WhileExit6
  %temp9 = alloca i32, align 4
  %39 = load i32, i32* %i, align 4
  %40 = load i32*, i32** %a, align 8
  %41 = getelementptr i32, i32* %40, i32 %39
  %42 = load i32, i32* %41, align 4
  store i32 %42, i32* %temp9, align 4
  %43 = load i32, i32* %j, align 4
  %44 = load i32*, i32** %a, align 8
  %45 = getelementptr i32, i32* %44, i32 %43
  %46 = load i32, i32* %45, align 4
  %47 = load i32, i32* %i, align 4
  %48 = load i32*, i32** %a, align 8
  %49 = getelementptr i32, i32* %48, i32 %47
  store i32 %46, i32* %49, align 4
  %50 = load i32, i32* %temp9, align 4
  %51 = load i32, i32* %j, align 4
  %52 = load i32*, i32** %a, align 8
  %53 = getelementptr i32, i32* %52, i32 %51
  store i32 %50, i32* %53, align 4
  %54 = load i32, i32* %i, align 4
  %55 = add i32 %54, 1
  store i32 %55, i32* %i, align 4
  %56 = load i32, i32* %j, align 4
  %57 = sub i32 %56, 1
  store i32 %57, i32* %j, align 4
  br label %BBExit
```

```
BBExit:                                  ; preds = %WhileExit6, %BBEntry8
  br label %WhileCmp

BBEntry10:                               ; preds = %WhileExit
  %58 = load i32*, i32** %a, align 8
  %59 = load i32, i32* %left, align 4
  %60 = load i32, i32* %j, align 4
  %61 = call i32 @quicksort(i32* %58, i32 %59, i32 %60)
  br label %BBExit11

BBExit11:                                ; preds = %WhileExit, %BBEntry10
  %62 = load i32, i32* %i, align 4
  %63 = load i32, i32* %right, align 4
  %64 = icmp slt i32 %62, %63
  br i1 %64, label %BBEntry12, label %BBExit13

BBEntry12:                               ; preds = %BBExit11
  %65 = load i32*, i32** %a, align 8
  %66 = load i32, i32* %i, align 4
  %67 = load i32, i32* %right, align 4
  %68 = call i32 @quicksort(i32* %65, i32 %66, i32 %67)
  br label %BBExit13

BBExit13:                                ; preds = %BBExit11, %BBEntry12
  ret i32 0
}

define i32 @main() {
BBEntry:
  %n = alloca i32, align 4
  store i8 0, i32* %n, align 1
  %temp = alloca i32, align 4
  store i8 0, i32* %temp, align 1
  %a = alloca [10000 x i32], align 4
  %0 = call i32 (i8*, ...) @scank(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @0, i32 0, i32 0), i32* %n)
  %i = alloca i32, align 4
  store i8 0, i32* %i, align 1
  store i32 0, i32* %i, align 4
  br label %ForCmp
```

```
ForCmp:                                  ; preds = %ForIter, %BBEntry
  %1 = load i32, i32* %i, align 4
  %2 = load i32, i32* %n, align 4
  %3 = icmp slt i32 %1, %2
  br i1 %3, label %BBEntry1, label %ForExit

ForIter:                                 ; preds = %BBEntry1
  %4 = load i32, i32* %i, align 4
  %5 = add i32 %4, 1
  store i32 %5, i32* %i, align 4
  br label %ForCmp

ForExit:                                 ; preds = %ForCmp
  %6 = bitcast [10000 x i32]* %a to i32*
  %7 = load i32, i32* %n, align 4
  %8 = sub i32 %7, 1
  %9 = call i32 @quicksort(i32* %6, i32 0, i32 %8)
  store i32 0, i32* %i, align 4
  br label %ForCmp2

BBEntry1:                                ; preds = %ForCmp
  %10 = call i32 (i8*, ...) @scank(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @1, i32 0, i32 0), i32* %temp)
  %11 = load i32, i32* %temp, align 4
  %12 = load i32, i32* %i, align 4
  %13 = bitcast [10000 x i32]* %a to i32*
  %14 = getelementptr i32, i32* %13, i32 %12
  store i32 %11, i32* %14, align 4
  br label %ForIter

ForCmp2:                                 ; preds = %ForIter3, %ForExit
  %15 = load i32, i32* %i, align 4
  %16 = load i32, i32* %n, align 4
  %17 = icmp slt i32 %15, %16
  br i1 %17, label %BBEntry5, label %ForExit4

ForIter3:                                ; preds = %BBEntry5
  %18 = load i32, i32* %i, align 4
  %19 = add i32 %18, 1
  store i32 %19, i32* %i, align 4
  br label %ForCmp2

ForExit4:                                ; preds = %ForCmp2
  ret i32 0

BBEntry5:                                ; preds = %ForCmp2
  %20 = load i32, i32* %i, align 4
  %21 = bitcast [10000 x i32]* %a to i32*
  %22 = getelementptr i32, i32* %21, i32 %20
  %23 = load i32, i32* %22, align 4
  %24 = call i32 (i32*, ...) @printk(i32* bitcast ([4 x i8]* @2 to i32*), i32 %23)
  br label %ForIter3
}
```

运行结果:



```
fixed case 0 (size [1x1]x[1x1])...pass!
fixed case 1 (size [1x1]x[2x1])...pass!
fixed case 2 (size [1x4]x[4x1])...pass!
fixed case 3 (size [4x1]x[1x4])...pass!
fixed case 4 (size [1x25]x[25x1])...pass!
randomly generated case 0 (size [20x20]x[20x20])...pass!
randomly generated case 1 (size [20x20]x[20x20])...pass!
randomly generated case 2 (size [20x20]x[20x20])...pass!
randomly generated case 3 (size [20x20]x[20x20])...pass!
randomly generated case 4 (size [20x20]x[20x20])...pass!
randomly generated case 5 (size [20x20]x[20x20])...pass!
randomly generated case 6 (size [20x20]x[20x20])...pass!
randomly generated case 7 (size [20x20]x[20x20])...pass!
randomly generated case 8 (size [20x20]x[20x20])...pass!
randomly generated case 9 (size [20x20]x[20x20])...pass!
----------------------------------------
2023-05-28 10:29:35.523
```

3.2 Matrix

测试代码:

```
static int c;
static int d[100];
int printk(int ptr, ...);
int scank(int ptr, ...);

int main(){
  int a2;
  int b1;
  int a1;
  int b2 =2;
  d[1] = b2;
  int Arr1[1000];
  int Arr2[1000];
  int Res[1000];
  int i;
  int j;
  int k;
  int temp1;



  scank("%d %d", &a1, &b1);

  for(i = 0;i<a1;i=i+1){
    for(j=0;j<b1;j=j+1){
      scank("%d", &temp1);
      Arr1[i*25+j] = temp1;
    }
  }

  scank("%d %d ", &a2, &b2);
  for(i = 0;i<a2;i=i+1){
    for(j=0;j<b2;j=j+1){
      scank("%d", &temp1);
      Arr2[i*25+j] = temp1;
    }
  }
  if(b1 != a2){
    printk("Incompatible Dimensions\n");
    return 0;
  }
  for(i = 0;i<a1;i=i+1){
    for(j=0;j<b2;j=j+1){
      int sum = 0;
      for(k=0;k<b1;k=k+1){
        int temp1 = Arr1[i*25+k];
        int temp2 = Arr2[k*25+j];
        sum = sum + temp1 * temp2;
      }
      Res[i*25+j] = sum;
    }
  }
  for(i=0;i<a1;i=i+1){
    for(j=0;j<b2;j=j+1){
      int temp = Res[i*25+j];
      printk("%10d",temp);
    }
    printk("\n");
  }

  return 0;
}
```

IR代码：

```
; ModuleID = 'main'
source_filename = "main"

@c = global i32 0
@d = global [100 x i32] undef
@0 = private unnamed_addr constant [6 x i8] c"%d %d\00", align 1
@1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@2 = private unnamed_addr constant [7 x i8] c"%d %d \00", align 1
@3 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@4 = private unnamed_addr constant [25 x i8] c"Incompatible Dimensions\0A\00", align 1
```

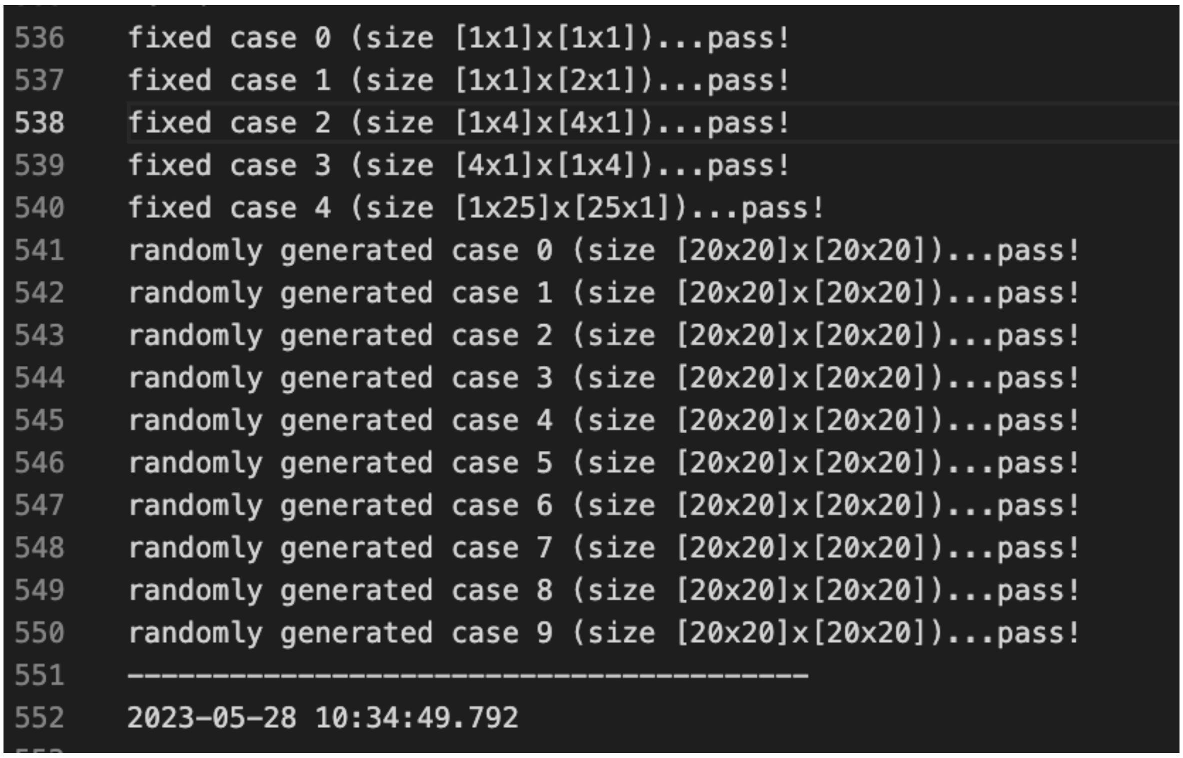
```
@5 = private unnamed_addr constant [5 x i8] c"%10d\00", align 1
@6 = private unnamed_addr constant [2 x i8] c"\0A\00", align 1

declare i32 @printk(i32*, ...)

declare i32 @scank(i32*, ...)

define i32 @main() {
BBEntry:
  %a2 = alloca i32, align 4
  store i8 0, i32* %a2, align 1
  %b1 = alloca i32, align 4
  store i8 0, i32* %b1, align 1
  %a1 = alloca i32, align 4
  store i8 0, i32* %a1, align 1
  %b2 = alloca i32, align 4
  store i32 2, i32* %b2, align 4
  %0 = load i32, i32* %b2, align 4
  store i32 %0, i32* getelementptr inbounds ([100 x i32], [100 x i32]* @d, i32 0, i32 1), align 4
  %Arr1 = alloca [1000 x i32], align 4
  %Arr2 = alloca [1000 x i32], align 4
  %Res = alloca [1000 x i32], align 4
  %i = alloca i32, align 4
  store i8 0, i32* %i, align 1
  %j = alloca i32, align 4
  store i8 0, i32* %j, align 1
  %k = alloca i32, align 4
  store i8 0, i32* %k, align 1
  %temp1 = alloca i32, align 4
  store i8 0, i32* %temp1, align 1
  %1 = call i32 (i32*, ...) @scank(i32* bitcast ([6 x i8]* @0 to i32*), i32* %a1, i32* %b1)
  store i32 0, i32* %i, align 4
  br label %ForCmp

ForCmp:                                           ; preds = %ForIter, %BBEntry
  %2 = load i32, i32* %i, align 4
  %3 = load i32, i32* %a1, align 4
  %4 = icmp slt i32 %2, %3
  br i1 %4, label %BBEntry1, label %ForExit
```

```
ForIter:                                    ; preds = %ForExit4
  %5 = load i32, i32* %i, align 4
  %6 = add i32 %5, 1
  store i32 %6, i32* %i, align 4
  br label %ForCmp

ForExit:                                    ; preds = %ForCmp
  %7 = call i32 (i32*, ...) @scank(i32* bitcast ([7 x i8]* @2 to i32*), i32* %a2, i32* %b2)
  store i32 0, i32* %i, align 4
  br label %ForCmp6

BBEntry1:                                   ; preds = %ForCmp
  store i32 0, i32* %j, align 4
  br label %ForCmp2

ForCmp2:                                    ; preds = %ForIter3, %BBEntry1
  %8 = load i32, i32* %j, align 4
  %9 = load i32, i32* %b1, align 4
  %10 = icmp slt i32 %8, %9
  br i1 %10, label %BBEntry5, label %ForExit4

ForIter3:                                   ; preds = %BBEntry5
  %11 = load i32, i32* %j, align 4
  %12 = add i32 %11, 1
  store i32 %12, i32* %j, align 4
  br label %ForCmp2

ForExit4:                                   ; preds = %ForCmp2
  br label %ForIter

BBEntry5:                                   ; preds = %ForCmp2
  %13 = call i32 (i32*, ...) @scank(i32* bitcast ([3 x i8]* @1 to i32*), i32* %temp1)
  %14 = load i32, i32* %temp1, align 4
  %15 = load i32, i32* %i, align 4
  %16 = mul i32 %15, 25
  %17 = load i32, i32* %j, align 4
  %18 = add i32 %16, %17
  %19 = bitcast [1000 x i32]* %Arr1 to i32*
  %20 = getelementptr i32, i32* %19, i32 %18
  store i32 %14, i32* %20, align 4
  br label %ForIter3

ForCmp6:                                          ; preds = %ForIter7, %ForExit
  %21 = load i32, i32* %i, align 4
  %22 = load i32, i32* %a2, align 4
  %23 = icmp slt i32 %21, %22
  br i1 %23, label %BBEntry9, label %ForExit8

ForIter7:                                         ; preds = %ForExit12
  %24 = load i32, i32* %i, align 4
  %25 = add i32 %24, 1
  store i32 %25, i32* %i, align 4
  br label %ForCmp6

ForExit8:                                         ; preds = %ForCmp6
  %26 = load i32, i32* %b1, align 4
  %27 = load i32, i32* %a2, align 4
  %28 = icmp ne i32 %26, %27
  br i1 %28, label %BBEntry14, label %BBExit

BBEntry9:                                         ; preds = %ForCmp6
  store i32 0, i32* %j, align 4
  br label %ForCmp10

ForCmp10:                                         ; preds = %ForIter11, %BBEntry9
  %29 = load i32, i32* %j, align 4
  %30 = load i32, i32* %b2, align 4
  %31 = icmp slt i32 %29, %30
  br i1 %31, label %BBEntry13, label %ForExit12

ForIter11:                                        ; preds = %BBEntry13
  %32 = load i32, i32* %j, align 4
  %33 = add i32 %32, 1
  store i32 %33, i32* %j, align 4
  br label %ForCmp10

ForExit12:                                        ; preds = %ForCmp10
```

```
  br label %ForIter7

BBEntry13:                                        ; preds = %ForCmp10
  %34 = call i32 (i32*, ...) @scank(i32* bitcast ([3 x i8]* @3 to i32*), i32* %temp1)
  %35 = load i32, i32* %temp1, align 4
  %36 = load i32, i32* %i, align 4
  %37 = mul i32 %36, 25
  %38 = load i32, i32* %j, align 4
  %39 = add i32 %37, %38
  %40 = bitcast [1000 x i32]* %Arr2 to i32*
  %41 = getelementptr i32, i32* %40, i32 %39
  store i32 %35, i32* %41, align 4
  br label %ForIter11

BBEntry14:                                        ; preds = %ForExit8
  %42 = call i32 (i32*, ...) @printk(i32* bitcast ([25 x i8]* @4 to i32*))
  ret i32 0
  br label %BBExit

BBExit:                                           ; preds = %ForExit8, %BBEntry14
  store i32 0, i32* %i, align 4
  br label %ForCmp15

ForCmp15:                                         ; preds = %ForIter16, %BBExit
  %43 = load i32, i32* %i, align 4
  %44 = load i32, i32* %a1, align 4
  %45 = icmp slt i32 %43, %44
  br i1 %45, label %BBEntry18, label %ForExit17

ForIter16:                                        ; preds = %ForExit21
  %46 = load i32, i32* %i, align 4
  %47 = add i32 %46, 1
  store i32 %47, i32* %i, align 4
  br label %ForCmp15

ForExit17:                                        ; preds = %ForCmp15
  store i32 0, i32* %i, align 4
  br label %ForCmp28
```

```
BBEntry18:                                    ; preds = %ForCmp15
  store i32 0, i32* %j, align 4
  br label %ForCmp19

ForCmp19:                                     ; preds = %ForIter20, %BBEntry18
  %48 = load i32, i32* %j, align 4
  %49 = load i32, i32* %b2, align 4
  %50 = icmp slt i32 %48, %49
  br i1 %50, label %BBEntry22, label %ForExit21

ForIter20:                                    ; preds = %ForExit25
  %51 = load i32, i32* %j, align 4
  %52 = add i32 %51, 1
  store i32 %52, i32* %j, align 4
  br label %ForCmp19

ForExit21:                                    ; preds = %ForCmp19
  br label %ForIter16

BBEntry22:                                    ; preds = %ForCmp19
  %sum = alloca i32, align 4
  store i32 0, i32* %sum, align 4
  store i32 0, i32* %k, align 4
  br label %ForCmp23

ForCmp23:                                     ; preds = %ForIter24, %BBEntry22
  %53 = load i32, i32* %k, align 4
  %54 = load i32, i32* %b1, align 4
  %55 = icmp slt i32 %53, %54
  br i1 %55, label %BBEntry26, label %ForExit25

ForIter24:                                    ; preds = %BBEntry26
  %56 = load i32, i32* %k, align 4
  %57 = add i32 %56, 1
  store i32 %57, i32* %k, align 4
  br label %ForCmp23

ForExit25:                                    ; preds = %ForCmp23
  %58 = load i32, i32* %sum, align 4
  %59 = load i32, i32* %i, align 4
  %60 = mul i32 %59, 25
  %61 = load i32, i32* %j, align 4
  %62 = add i32 %60, %61
  %63 = bitcast [1000 x i32]* %Res to i32*
  %64 = getelementptr i32, i32* %63, i32 %62
  store i32 %58, i32* %64, align 4
  br label %ForIter20

BBEntry26:                                        ; preds = %ForCmp23
  %temp127 = alloca i32, align 4
  %65 = load i32, i32* %i, align 4
  %66 = mul i32 %65, 25
  %67 = load i32, i32* %k, align 4
  %68 = add i32 %66, %67
  %69 = bitcast [1000 x i32]* %Arr1 to i32*
  %70 = getelementptr i32, i32* %69, i32 %68
  %71 = load i32, i32* %70, align 4
  store i32 %71, i32* %temp127, align 4
  %temp2 = alloca i32, align 4
  %72 = load i32, i32* %k, align 4
  %73 = mul i32 %72, 25
  %74 = load i32, i32* %j, align 4
  %75 = add i32 %73, %74
  %76 = bitcast [1000 x i32]* %Arr2 to i32*
  %77 = getelementptr i32, i32* %76, i32 %75
  %78 = load i32, i32* %77, align 4
  store i32 %78, i32* %temp2, align 4
  %79 = load i32, i32* %sum, align 4
  %80 = load i32, i32* %temp127, align 4
  %81 = load i32, i32* %temp2, align 4
  %82 = mul i32 %80, %81
  %83 = add i32 %79, %82
  store i32 %83, i32* %sum, align 4
  br label %ForIter24

ForCmp28:                                         ; preds = %ForIter29, %ForExit17
  %84 = load i32, i32* %i, align 4
  %85 = load i32, i32* %a1, align 4
  %86 = icmp slt i32 %84, %85
  br i1 %86, label %BBEntry31, label %ForExit30

ForIter29:                              ; preds = %ForExit34
  %87 = load i32, i32* %i, align 4
  %88 = add i32 %87, 1
  store i32 %88, i32* %i, align 4
  br label %ForCmp28

ForExit30:                              ; preds = %ForCmp28
  ret i32 0

BBEntry31:                              ; preds = %ForCmp28
  store i32 0, i32* %j, align 4
  br label %ForCmp32

ForCmp32:                               ; preds = %ForIter33, %BBEntry31
  %89 = load i32, i32* %j, align 4
  %90 = load i32, i32* %b2, align 4
  %91 = icmp slt i32 %89, %90
  br i1 %91, label %BBEntry35, label %ForExit34

ForIter33:                              ; preds = %BBEntry35
  %92 = load i32, i32* %j, align 4
  %93 = add i32 %92, 1
  store i32 %93, i32* %j, align 4
  br label %ForCmp32

ForExit34:                              ; preds = %ForCmp32
  %94 = call i32 (i32*, ...) @printk(i32* bitcast ([2 x i8]* @6 to i32*))
  br label %ForIter29

BBEntry35:                              ; preds = %ForCmp32
  %temp = alloca i32, align 4
  %95 = load i32, i32* %i, align 4
  %96 = mul i32 %95, 25
  %97 = load i32, i32* %j, align 4
  %98 = add i32 %96, %97
  %99 = bitcast [1000 x i32]* %Res to i32*
  %100 = getelementptr i32, i32* %99, i32 %98
  %101 = load i32, i32* %100, align 4
  store i32 %101, i32* %temp, align 4
  %102 = load i32, i32* %temp, align 4
  %103 = call i32 (i32*, ...) @printk(i32* bitcast ([5 x i8]* @5 to i32*), i32 %102)
  br label %ForIter33
}
```

运行结果:



```
fixed case 0 (size [1x1]x[1x1])...pass!
fixed case 1 (size [1x1]x[2x1])...pass!
fixed case 2 (size [1x4]x[4x1])...pass!
fixed case 3 (size [4x1]x[1x4])...pass!
fixed case 4 (size [1x25]x[25x1])...pass!
randomly generated case 0 (size [20x20]x[20x20])...pass!
randomly generated case 1 (size [20x20]x[20x20])...pass!
randomly generated case 2 (size [20x20]x[20x20])...pass!
randomly generated case 3 (size [20x20]x[20x20])...pass!
randomly generated case 4 (size [20x20]x[20x20])...pass!
randomly generated case 5 (size [20x20]x[20x20])...pass!
randomly generated case 6 (size [20x20]x[20x20])...pass!
randomly generated case 7 (size [20x20]x[20x20])...pass!
randomly generated case 8 (size [20x20]x[20x20])...pass!
randomly generated case 9 (size [20x20]x[20x20])...pass!
----------------------------------------
2023-05-28 10:34:49.792
```

3.3 Advisor

测试代码:

```
int printk(int ptr, ...);
int scank(char ptr, ...);

static char strend = 0;
static char strnewline = 10;
static char strseq = 124;
static char strspace = 32;
static char strcomma = 44;
static char strsemicolon = 59;
static char str0 = 48;
static char str1 = 49;
static char str2 = 50;
static char str3 = 51;
static char str4 = 52;
static char str5 = 53;
static char gradeA = 65;
static char gradeB = 66;
static char gradeC = 67;
static char gradeD = 68;
static char gradeE = 69;
static char gradeF = 70;

int myStrlen(char ptr s) {
    int i;
    for (i = 0; s[i] != strend; i=i+1){}
    return i;
}

int charAddStr(char chr, char ptr s) {
    int i;
    i = myStrlen(s) + 1;
    for (; i>0; i=i-1) {
        s[i] = s[i-1];
    }
    s[0] = chr;

    return 0;
}

int charLeaveStr(char ptr s) {
    int i;
    int len = myStrlen(s);
    for (i = 0; i < len; i=i+1) {
        s[i] = s[i+1];
    }
    return 0;
}

int eqStr(char ptr s1, char ptr s2) {
    int i;
    for (i = 0; s1[i] != strend || s2[i] != strend; i=i+1){
        if (s1[i] != s2[i]) {
            return 0;
        }
    }
    return 1;
}

int copyStr(char ptr s1, char ptr s2, int begin, char seq) {
    int i = 0;
    for (i = 0; s2[begin] != seq; i=i+1) {
        s1[i] = s2[begin];
        begin = begin+1;
    }
    s1[i] = strend;
    return 0;
}

int appendStr(char ptr s1, char ptr s2, char seq) {
    int i;
    int len = myStrlen(s1);
    s1[len] = seq;
    len = len + 1;
    for(i = 0; s2[i] != strend; i=i+1) {
        s1[len] = s2[i];
        len = len+1;
    }
    s1[len] = strend;
    return 0;
}

int getSubStr(char ptr substr, char ptr str, int index, char seq) {
    int begin = 0;
    for (;index != 0; index = index-1){
        while(str[begin] != seq) {
            begin = begin+1;
        }
        begin = begin+1;
    }
    int i = 0;
    while(str[begin] != seq && str[begin] != strend) {
        substr[i] = str[begin];
        i = i+1;
        begin = begin+1;
    }
    substr[i] = strend;
    return 0;
}

int getSubCnt(char ptr str, char seq) {
    int index;
    int cnt = 1;
    if (str[0] == strend) {
        return 0;
    }
    for (index = 0; str[index] != strend; index = index+1){
        if (str[index] == seq) {
            cnt = cnt+1;
        }
    }
    return cnt;
}

int getStrIndex(char ptr substr, char ptr str, char seq) {
    int i = 0;
    int i2 = 0;
    int index = 0;
    int temp = 0;
    while(str[temp] != strend) {
        i = 0;
        while(substr[i] != strend && str[i2] != strend && str[i2] != seq && (str[i2] == substr[i])) {
            i2 = i2+1;
            i = i+1;
        }
        // printk("%d %d %d\n", index, i2, i);
        index = index + 1;
        if (substr[i] == strend && (str[i2] == strend || str[i2] == seq)){
            return index;
        }
        // remember to go to strend or seq
        while(str[i2] != strend && str[i2] != seq) {
            i2 = i2+1;
        }

        temp = i2;
        i2 = i2 + 1;
    }
    return 0;
}

int charToInt(char chr) {
    if (chr == str0) {
        return 0;
    }
    if (chr == str1) {
        return 1;
    }
    if (chr == str2) {
        return 2;
    }
    if (chr == str3) {
        return 3;
    }
    if (chr == str4) {
        return 4;
    }
    if (chr == str5) {
        return 5;
    }
    return 0;
}
```

```
int gradeToInt(char chr) {
    if (chr == gradeA) {
        return 4;
    }
    if (chr == gradeB) {
        return 3;
    }
    if (chr == gradeC) {
        return 2;
    }
    if (chr == gradeD) {
        return 1;
    }
    if (chr == gradeF) {
        return 0;
    }
    return 0;
}

int main(){
    char coursename[6];
    char coursescore;
    char coursepre[1000];
    char coursegrade;

    char oneline[1000];
    char firstchar;
    char tempchar;

    int score;
    int grade;

    int score_learn = 0;
    int score_get = 0;
    int score_remain = 0;
    int gpa = 0;
    int cnt = 0;
```

```
char names[1000];
char pres[1000000];
int learned[100];
int learnable[100];

int i;
int j;
int index;
int cursor;
int orCnt;
int andCnt;
int learn;
char orStr[100];
char andStr[6];

// get read
while(1) {
    // read the line
    scank("%c", &firstchar);
    if (firstchar == strnewline) {
        charLeaveStr(names);
        charLeaveStr(pres);
        // printk("%s\n", names);
        // printk("%s\n", pres);

        // read over
        for(cursor = 0; cursor < cnt; cursor = cursor+1) {
            if (learned[cursor]) {
                learnable[cursor] = 0;
            }
            else {
                learnable[cursor] = 0;
                getSubStr(coursepre, pres, cursor, strspace);
                orCnt = getSubCnt(coursepre, strsemicolon);
                if (orCnt == 0) {
                    learnable[cursor] = 1;
                }
                for (i = 0; i < orCnt && learnable[cursor]==0; i=i+1) {
                    learn = 1;
            getSubStr(orStr, coursepre, i, strsemicolon);
            andCnt = getSubCnt(orStr, strcomma);
            for (j = 0; j < andCnt && learn == 1; j=j+1) {
                // find out the course name, stored in andStr
                getSubStr(andStr, orStr, j, strcomma);

                index = getStrIndex(andStr, names, strspace);
                // printk("%d %d %s\n", cursor, index, andStr);
                if (index == 0) {
                    learn = 0;
                }
                else {
                    index = index-1;
                    if (learned[index] == 0) {
                        learn = 0;
                    }
                }
            }
            // can learn
            learnable[cursor] = learn;
        }
    }
}

// print the result
if (score_learn != 0){
    printk("GPA: %.1f\n", (1.0*gpa)/score_learn);
}
else {
    printk("GPA: 0.0\n");
}
printk("Hours Attempted: %d\n", score_learn);
printk("Hours Completed: %d\n", score_get);
printk("Credits Remaining: %d\n\n", score_remain);

// print recommended course
printk("Possible Courses to Take Next\n");
if (score_remain != 0) {
    for (cursor = 0; cursor < cnt; cursor = cursor+1) {
            if (learnable[cursor] == 1) {
                getSubStr(coursename, names, cursor, strspace);
                printk("  %s\n", coursename);
            }
        }
    }
    else {
        printk("  None - Congratulations!\n");
    }

    return 0;
}
scank("%s", oneline);
scank("%c", &tempchar); // remove the newline \n
charAddStr(firstchar, oneline);

// decode the line
cursor = 0;
copyStr(coursename, oneline, cursor, strseq);
cursor = myStrlen(coursename)+1;
coursescore = oneline[cursor];
cursor = cursor + 2;
copyStr(coursepre, oneline, cursor, strseq);
cursor = cursor + myStrlen(coursepre) + 1;
coursegrade = oneline[cursor];

score = charToInt(coursescore);
grade = gradeToInt(coursegrade);

// printk("%d %d %s %s\n", score, grade, coursename, coursepre);

// renew the statistical information
if (coursegrade == gradeF) {
    score_learn = score_learn + score;
    score_remain = score_remain + score;
    learned[cnt] = 0;
}
else {
    if (coursegrade == strend) {
```

```
                score_remain = score_remain + score;
                learned[cnt] = 0;
            }
            else {
                score_learn = score_learn + score;
                score_get = score_get + score;
                gpa = gpa + grade*score;
                learned[cnt] = 1;
            }
        }
        // record the course
        cnt = cnt + 1;
        appendStr(names, coursename, strspace);
        appendStr(pres, coursepre, strspace);

    }

  return 0;
}
```

IR 代码

```
; ModuleID = 'main'
source_filename = "main"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@strend = global i8 0
@strnewline = global i8 10
@strseq = global i8 124
@strspace = global i8 32
@strcomma = global i8 44
@strsemicolon = global i8 59
@str0 = global i8 48
@str1 = global i8 49
@str2 = global i8 50
@str3 = global i8 51
@str4 = global i8 52
```

```
@str5 = global i8 53
@gradeA = global i8 65
@gradeB = global i8 66
@gradeC = global i8 67
@gradeD = global i8 68
@gradeE = global i8 69
@gradeF = global i8 70
@0 = private unnamed_addr constant [3 x i8] c"%c\00", align 1
@1 = private unnamed_addr constant [11 x i8] c"GPA: %.1f\0A\00", align 1
@2 = private unnamed_addr constant [10 x i8] c"GPA: 0.0\0A\00", align 1
@3 = private unnamed_addr constant [21 x i8] c"Hours Attempted: %d\0A\00", align 1
@4 = private unnamed_addr constant [21 x i8] c"Hours Completed: %d\0A\00", align 1
@5 = private unnamed_addr constant [24 x i8] c"Credits Remaining: %d\0A\0A\00", align 1
@6 = private unnamed_addr constant [31 x i8] c"Possible Courses to Take Next\0A\00", align 1
@7 = private unnamed_addr constant [6 x i8] c"  %s\0A\00", align 1
@8 = private unnamed_addr constant [27 x i8] c"  None - Congratulations!\0A\00", align 1
@9 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@10 = private unnamed_addr constant [3 x i8] c"%c\00", align 1

declare i32 @printk(i32*, ...)

declare i32 @scank(i8*, ...)

define i32 @myStrlen(i8* %0) {
BBEntry:
  %s = alloca i8*, align 8
  store i8* %0, i8** %s, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  store i32 0, i32* %i, align 4
  br label %ForCmp

ForCmp:                                           ; preds = %BBEntry1, %BBEntry
  %1 = load i32, i32* %i, align 4
  %2 = load i8*, i8** %s, align 8
  %3 = getelementptr i8, i8* %2, i32 %1
  %4 = load i8, i8* %3, align 1
  %5 = load i8, i8* @strend, align 1
  %6 = icmp ne i8 %4, %5
  br i1 %6, label %BBEntry1, label %ForExit

ForExit:                                        ; preds = %ForCmp
  %7 = load i32, i32* %i, align 4
  ret i32 %7

BBEntry1:                                       ; preds = %ForCmp
  %8 = load i32, i32* %i, align 4
  %9 = add i32 %8, 1
  store i32 %9, i32* %i, align 4
  br label %ForCmp
}

define i32 @charAddStr(i8 %0, i8* %1) {
BBEntry:
  %s = alloca i8*, align 8
  store i8* %1, i8** %s, align 8
  %chr = alloca i8, align 1
  store i8 %0, i8* %chr, align 1
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  %2 = load i8*, i8** %s, align 8
  %3 = call i32 @myStrlen(i8* %2)
  %4 = add i32 %3, 1
  store i32 %4, i32* %i, align 4
  br label %ForCmp

ForCmp:                                         ; preds = %BBEntry1, %BBEntry
  %5 = load i32, i32* %i, align 4
  %6 = icmp sgt i32 %5, 0
  br i1 %6, label %BBEntry1, label %ForExit

ForExit:                                        ; preds = %ForCmp
  %7 = load i8, i8* %chr, align 1
  %8 = load i8*, i8 ** %s, align 8
  %9 = bitcast i8* %8 to i8*
  store i8 %7, i8* %9, align 1
  ret i32 0
```

```
BBEntry1:                                        ; preds = %ForCmp
  %10 = load i32, i32* %i, align 4
  %11 = sub i32 %10, 1
  %12 = load i8*, i8** %s, align 8
  %13 = getelementptr i8, i8* %12, i32 %11
  %14 = load i8, i8* %13, align 1
  %15 = load i32, i32* %i, align 4
  %16 = load i8*, i8** %s, align 8
  %17 = getelementptr i8, i8* %16, i32 %15
  store i8 %14, i8* %17, align 1
  %18 = load i32, i32* %i, align 4
  %19 = sub i32 %18, 1
  store i32 %19, i32* %i, align 4
  br label %ForCmp
}

define i32 @charLeaveStr(i8* %0) {
BBEntry:
  %s = alloca i8*, align 8
  store i8* %0, i8** %s, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  %len = alloca i32, align 4
  %1 = load i8*, i8** %s, align 8
  %2 = call i32 @myStrlen(i8* %1)
  store i32 %2, i32* %len, align 4
  store i32 0, i32* %i, align 4
  br label %ForCmp

ForCmp:                                          ; preds = %ForIter, %BBEntry
  %3 = load i32, i32* %i, align 4
  %4 = load i32, i32* %len, align 4
  %5 = icmp slt i32 %3, %4
  br i1 %5, label %BBEntry1, label %ForExit

ForIter:                                         ; preds = %BBEntry1
  %6 = load i32, i32* %i, align 4
  %7 = add i32 %6, 1
  store i32 %7, i32* %i, align 4
  br label %ForCmp

ForExit:                                          ; preds = %ForCmp
  ret i32 0

BBEntry1:                                         ; preds = %ForCmp
  %8 = load i32, i32* %i, align 4
  %9 = add i32 %8, 1
  %10 = load i8*, i8** %s, align 8
  %11 = getelementptr i8, i8* %10, i32 %9
  %12 = load i8, i8* %11, align 1
  %13 = load i32, i32* %i, align 4
  %14 = load i8*, i8** %s, align 8
  %15 = getelementptr i8, i8* %14, i32 %13
  store i8 %12, i8* %15, align 1
  br label %ForIter
}

define i32 @eqStr(i8* %0, i8* %1) {
BBEntry:
  %s2 = alloca i8*, align 8
  store i8* %1, i8** %s2, align 8
  %s1 = alloca i8*, align 8
  store i8* %0, i8** %s1, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  store i32 0, i32* %i, align 4
  br label %ForCmp

ForCmp:                                           ; preds = %BBExit, %BBEntry
  %2 = load i32, i32* %i, align 4
  %3 = load i8*, i8** %s1, align 8
  %4 = getelementptr i8, i8* %3, i32 %2
  %5 = load i8, i8* %4, align 1
  %6 = load i8, i8* @strend, align 1
  %7 = icmp ne i8 %5, %6
  %8 = load i32, i32* %i, align 4
  %9 = load i8*, i8** %s2, align 8
  %10 = getelementptr i8, i8* %9, i32 %8
  %11 = load i8, i8* %10, align 1
  %12 = load i8, i8* @strend, align 1
  %13 = icmp ne i8 %11, %12
  %14 = select i1 %7, i1 true, i1 %13
  br i1 %14, label %BBEntry1, label %ForExit

ForExit:                                          ; preds = %ForCmp
  ret i32 1

BBEntry1:                                         ; preds = %ForCmp
  %15 = load i32, i32* %i, align 4
  %16 = load i8*, i8** %s1, align 8
  %17 = getelementptr i8, i8* %16, i32 %15
  %18 = load i8, i8* %17, align 1
  %19 = load i32, i32* %i, align 4
  %20 = load i8*, i8** %s2, align 8
  %21 = getelementptr i8, i8* %20, i32 %19
  %22 = load i8, i8* %21, align 1
  %23 = icmp ne i8 %18, %22
  br i1 %23, label %BBEntry2, label %BBExit

BBEntry2:                                         ; preds = %BBEntry1
  ret i32 0
  br label %BBExit

BBExit:                                           ; preds = %BBEntry1, %BBEntry2
  %24 = load i32, i32* %i, align 4
  %25 = add i32 %24, 1
  store i32 %25, i32* %i, align 4
  br label %ForCmp
}

define i32 @copyStr(i8* %0, i8* %1, i32 %2, i8 %3) {
BBEntry:
  %seq = alloca i8, align 1
  store i8 %3, i8* %seq, align 1
  %begin = alloca i32, align 4
  store i32 %2, i32* %begin, align 4
  %s2 = alloca i8*, align 8
  store i8* %1, i8** %s2, align 8
  %s1 = alloca i8*, align 8
  store i8* %0, i8** %s1, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  store i32 0, i32* %i, align 4
  br label %ForCmp

ForCmp:                                          ; preds = %ForIter, %BBEntry
  %4 = load i32, i32* %begin, align 4
  %5 = load i8*, i8** %s2, align 8
  %6 = getelementptr i8, i8* %5, i32 %4
  %7 = load i8, i8* %6, align 1
  %8 = load i8, i8* %seq, align 1
  %9 = icmp ne i8 %7, %8
  br i1 %9, label %BBEntry1, label %ForExit

ForIter:                                         ; preds = %BBEntry1
  %10 = load i32, i32* %i, align 4
  %11 = add i32 %10, 1
  store i32 %11, i32* %i, align 4
  br label %ForCmp

ForExit:                                         ; preds = %ForCmp
  %12 = load i8, i8* @strend, align 1
  %13 = load i32, i32* %i, align 4
  %14 = load i8*, i8** %s1, align 8
  %15 = getelementptr i8, i8* %14, i32 %13
  store i8 %12, i8* %15, align 1
  ret i32 0

BBEntry1:                                        ; preds = %ForCmp
  %16 = load i32, i32* %begin, align 4
  %17 = load i8*, i8** %s2, align 8
  %18 = getelementptr i8, i8* %17, i32 %16
  %19 = load i8, i8* %18, align 1
  %20 = load i32, i32* %i, align 4
  %21 = load i8*, i8** %s1, align 8
  %22 = getelementptr i8, i8* %21, i32 %20
  store i8 %19, i8* %22, align 1
  %23 = load i32, i32* %begin, align 4
  %24 = add i32 %23, 1
  store i32 %24, i32* %begin, align 4
  br label %ForIter
}

define i32 @appendStr(i8* %0, i8* %1, i8 %2) {
BBEntry:
  %seq = alloca i8, align 1
  store i8 %2, i8* %seq, align 1
  %s2 = alloca i8*, align 8
  store i8* %1, i8** %s2, align 8
  %s1 = alloca i8*, align 8
  store i8* %0, i8** %s1, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  %len = alloca i32, align 4
  %3 = load i8*, i8** %s1, align 8
  %4 = call i32 @myStrlen(i8* %3)
  store i32 %4, i32* %len, align 4
  %5 = load i8, i8* %seq, align 1
  %6 = load i32, i32* %len, align 4
  %7 = load i8*, i8** %s1, align 8
  %8 = getelementptr i8, i8* %7, i32 %6
  store i8 %5, i8* %8, align 1
  %9 = load i32, i32* %len, align 4
  %10 = add i32 %9, 1
  store i32 %10, i32* %len, align 4
  store i32 0, i32* %i, align 4
  br label %ForCmp

ForCmp:                                           ; preds = %ForIter, %BBEntry
  %11 = load i32, i32* %i, align 4
  %12 = load i8*, i8** %s2, align 8
  %13 = getelementptr i8, i8* %12, i32 %11
  %14 = load i8, i8* %13, align 1
  %15 = load i8, i8* @strend, align 1
  %16 = icmp ne i8 %14, %15
  br i1 %16, label %BBEntry1, label %ForExit

ForIter:                                          ; preds = %BBEntry1
  %17 = load i32, i32* %i, align 4
  %18 = add i32 %17, 1
  store i32 %18, i32* %i, align 4
  br label %ForCmp

ForExit:                                          ; preds = %ForCmp
  %19 = load i8, i8* @strend, align 1
  %20 = load i32, i32* %len, align 4
  %21 = load i8*, i8** %s1, align 8
  %22 = getelementptr i8, i8* %21, i32 %20
  store i8 %19, i8* %22, align 1
  ret i32 0

BBEntry1:                                         ; preds = %ForCmp
  %23 = load i32, i32* %i, align 4
  %24 = load i8*, i8** %s2, align 8
  %25 = getelementptr i8, i8* %24, i32 %23
  %26 = load i8, i8* %25, align 1
  %27 = load i32, i32* %len, align 4
  %28 = load i8*, i8** %s1, align 8
  %29 = getelementptr i8, i8* %28, i32 %27
  store i8 %26, i8* %29, align 1
  %30 = load i32, i32* %len, align 4
  %31 = add i32 %30, 1
  store i32 %31, i32* %len, align 4
  br label %ForIter
}

define i32 @getSubStr(i8* %0, i8* %1, i32 %2, i8 %3) {
BBEntry:
  %seq = alloca i8, align 1
  store i8 %3, i8* %seq, align 1
  %index = alloca i32, align 4
  store i32 %2, i32* %index, align 4
  %str = alloca i8*, align 8
  store i8* %1, i8** %str, align 8
  %substr = alloca i8*, align 8
  store i8* %0, i8** %substr, align 8
  %begin = alloca i32, align 4
  store i32 0, i32* %begin, align 4
  br label %ForCmp

ForCmp:                                           ; preds = %ForIter, %BBEntry
  %4 = load i32, i32* %index, align 4
  %5 = icmp ne i32 %4, 0
  br i1 %5, label %BBEntry1, label %ForExit

ForIter:                                          ; preds = %WhileExit
  %6 = load i32, i32* %index, align 4
  %7 = sub i32 %6, 1
  store i32 %7, i32* %index, align 4
  br label %ForCmp

ForExit:                                          ; preds = %ForCmp
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  br label %WhileCmp3

BBEntry1:                                         ; preds = %ForCmp
  br label %WhileCmp

WhileCmp:                                         ; preds = %BBEntry2, %BBEntry1
  %8 = load i32, i32* %begin, align 4
  %9 = load i8*, i8** %str, align 8
  %10 = getelementptr i8, i8* %9, i32 %8
  %11 = load i8, i8* %10, align 1
  %12 = load i8, i8* %seq, align 1
  %13 = icmp ne i8 %11, %12
  br i1 %13, label %BBEntry2, label %WhileExit

WhileExit:                                        ; preds = %WhileCmp
  %14 = load i32, i32* %begin, align 4
  %15 = add i32 %14, 1
  store i32 %15, i32* %begin, align 4
  br label %ForIter
```

```
BBEntry2:                                       ; preds = %WhileCmp
  %16 = load i32, i32* %begin, align 4
  %17 = add i32 %16, 1
  store i32 %17, i32* %begin, align 4
  br label %WhileCmp

WhileCmp3:                                      ; preds = %BBEntry5, %ForExit
  %18 = load i32, i32* %begin, align 4
  %19 = load i8*, i8** %str, align 8
  %20 = getelementptr i8, i8* %19, i32 %18
  %21 = load i8, i8* %20, align 1
  %22 = load i8, i8* %seq, align 1
  %23 = icmp ne i8 %21, %22
  %24 = load i32, i32* %begin, align 4
  %25 = load i8*, i8** %str, align 8
  %26 = getelementptr i8, i8* %25, i32 %24
  %27 = load i8, i8* %26, align 1
  %28 = load i8, i8* @strend, align 1
  %29 = icmp ne i8 %27, %28
  %30 = select i1 %23, i1 %29, i1 false
  br i1 %30, label %BBEntry5, label %WhileExit4

WhileExit4:                                     ; preds = %WhileCmp3
  %31 = load i8, i8* @strend, align 1
  %32 = load i32, i32* %i, align 4
  %33 = load i8*, i8** %substr, align 8
  %34 = getelementptr i8, i8* %33, i32 %32
  store i8 %31, i8* %34, align 1
  ret i32 0

BBEntry5:                                       ; preds = %WhileCmp3
  %35 = load i32, i32* %begin, align 4
  %36 = load i8*, i8** %str, align 8
  %37 = getelementptr i8, i8* %36, i32 %35
  %38 = load i8, i8* %37, align 1
  %39 = load i32, i32* %i, align 4
  %40 = load i8*, i8** %substr, align 8
  %41 = getelementptr i8, i8* %40, i32 %39
  store i8 %38, i8* %41, align 1
  %42 = load i32, i32* %i, align 4
  %43 = add i32 %42, 1
  store i32 %43, i32* %i, align 4
  %44 = load i32, i32* %begin, align 4
  %45 = add i32 %44, 1
  store i32 %45, i32* %begin, align 4
  br label %WhileCmp3
}

define i32 @getSubCnt(i8* %0, i8 %1) {
BBEntry:
  %seq = alloca i8, align 1
  store i8 %1, i8* %seq, align 1
  %str = alloca i8*, align 8
  store i8* %0, i8** %str, align 8
  %index = alloca i32, align 4
  store i32 0, i32* %index, align 4
  %cnt = alloca i32, align 4
  store i32 1, i32* %cnt, align 4
  %2 = load i8*, i8** %str, align 8
  %3 = bitcast i8* %2 to i8*
  %4 = load i8, i8* %3, align 1
  %5 = load i8, i8* @strend, align 1
  %6 = icmp eq i8 %4, %5
  br i1 %6, label %BBEntry1, label %BBExit

BBEntry1:                                         ; preds = %BBEntry
  ret i32 0
  br label %BBExit

BBExit:                                           ; preds = %BBEntry, %BBEntry1
  store i32 0, i32* %index, align 4
  br label %ForCmp

ForCmp:                                           ; preds = %BBExit4, %BBExit
  %7 = load i32, i32* %index, align 4
  %8 = load i8*, i8** %str, align 8
  %9 = getelementptr i8, i8* %8, i32 %7
  %10 = load i8, i8* %9, align 1
  %11 = load i8, i8* @strend, align 1
  %12 = icmp ne i8 %10, %11
  br i1 %12, label %BBEntry2, label %ForExit

ForExit:                                          ; preds = %ForCmp
  %13 = load i32, i32* %cnt, align 4
  ret i32 %13

BBEntry2:                                         ; preds = %ForCmp
  %14 = load i32, i32* %index, align 4
  %15 = load i8*, i8** %str, align 8
  %16 = getelementptr i8, i8* %15, i32 %14
  %17 = load i8, i8* %16, align 1
  %18 = load i8, i8* %seq, align 1
  %19 = icmp eq i8 %17, %18
  br i1 %19, label %BBEntry3, label %BBExit4

BBEntry3:                                         ; preds = %BBEntry2
  %20 = load i32, i32* %cnt, align 4
  %21 = add i32 %20, 1
  store i32 %21, i32* %cnt, align 4
  br label %BBExit4

BBExit4:                                          ; preds = %BBEntry2, %BBEntry3
  %22 = load i32, i32* %index, align 4
  %23 = add i32 %22, 1
  store i32 %23, i32* %index, align 4
  br label %ForCmp
}

define i32 @getStrIndex(i8* %0, i8* %1, i8 %2) {
BBEntry:
  %seq = alloca i8, align 1
  store i8 %2, i8* %seq, align 1
  %str = alloca i8*, align 8
  store i8* %1, i8** %str, align 8
  %substr = alloca i8*, align 8
  store i8* %0, i8** %substr, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  %i2 = alloca i32, align 4
  store i32 0, i32* %i2, align 4
  %index = alloca i32, align 4
  store i32 0, i32* %index, align 4
  %temp = alloca i32, align 4
  store i32 0, i32* %temp, align 4
  br label %WhileCmp

WhileCmp:                                         ; preds = %WhileExit7, %BBEntry
  %3 = load i32, i32* %temp, align 4
  %4 = load i8*, i8** %str, align 8
  %5 = getelementptr i8, i8* %4, i32 %3
  %6 = load i8, i8* %5, align 1
  %7 = load i8, i8* @strend, align 1
  %8 = icmp ne i8 %6, %7
  br i1 %8, label %BBEntry1, label %WhileExit

WhileExit:                                        ; preds = %WhileCmp
  ret i32 0

BBEntry1:                                         ; preds = %WhileCmp
  store i32 0, i32* %i, align 4
  br label %WhileCmp2

WhileCmp2:                                        ; preds = %BBEntry4, %BBEntry1
  %9 = load i32, i32* %i, align 4
  %10 = load i8*, i8** %substr, align 8
  %11 = getelementptr i8, i8* %10, i32 %9
  %12 = load i8, i8* %11, align 1
  %13 = load i8, i8* @strend, align 1
  %14 = icmp ne i8 %12, %13
  %15 = load i32, i32* %i2, align 4
  %16 = load i8*, i8** %str, align 8
  %17 = getelementptr i8, i8* %16, i32 %15
  %18 = load i8, i8* %17, align 1
  %19 = load i8, i8* @strend, align 1
  %20 = icmp ne i8 %18, %19
  %21 = select i1 %14, i1 %20, i1 false
  %22 = load i32, i32* %i2, align 4
  %23 = load i8*, i8** %str, align 8
  %24 = getelementptr i8, i8* %23, i32 %22
  %25 = load i8, i8* %24, align 1
  %26 = load i8, i8* %seq, align 1
  %27 = icmp ne i8 %25, %26
  %28 = select i1 %21, i1 %27, i1 false
  %29 = load i32, i32* %i2, align 4
  %30 = load i8*, i8** %str, align 8
  %31 = getelementptr i8, i8* %30, i32 %29
  %32 = load i8, i8* %31, align 1
  %33 = load i32, i32* %i, align 4
  %34 = load i8*, i8** %substr, align 8
  %35 = getelementptr i8, i8* %34, i32 %33
  %36 = load i8, i8* %35, align 1
  %37 = icmp eq i8 %32, %36
  %38 = select i1 %28, i1 %37, i1 false
  br i1 %38, label %BBEntry4, label %WhileExit3

WhileExit3:                                       ; preds = %WhileCmp2
  %39 = load i32, i32* %index, align 4
  %40 = add i32 %39, 1
  store i32 %40, i32* %index, align 4
  %41 = load i32, i32* %i, align 4
  %42 = load i8*, i8** %substr, align 8
  %43 = getelementptr i8, i8* %42, i32 %41
  %44 = load i8, i8* %43, align 1
  %45 = load i8, i8* @strend, align 1
  %46 = icmp eq i8 %44, %45
  %47 = load i32, i32* %i2, align 4
  %48 = load i8*, i8** %str, align 8
  %49 = getelementptr i8, i8* %48, i32 %47
  %50 = load i8, i8* %49, align 1
  %51 = load i8, i8* @strend, align 1
  %52 = icmp eq i8 %50, %51
  %53 = load i32, i32* %i2, align 4
  %54 = load i8*, i8** %str, align 8
  %55 = getelementptr i8, i8* %54, i32 %53
  %56 = load i8, i8* %55, align 1
  %57 = load i8, i8* %seq, align 1
  %58 = icmp eq i8 %56, %57
  %59 = select i1 %52, i1 true, i1 %58
  %60 = select i1 %46, i1 %59, i1 false
  br i1 %60, label %BBEntry5, label %BBExit

BBEntry4:                                    ; preds = %WhileCmp2
  %61 = load i32, i32* %i2, align 4
  %62 = add i32 %61, 1
  store i32 %62, i32* %i2, align 4
  %63 = load i32, i32* %i, align 4
  %64 = add i32 %63, 1
  store i32 %64, i32* %i, align 4
  br label %WhileCmp2

BBEntry5:                                    ; preds = %WhileExit3
  %65 = load i32, i32* %index, align 4
  ret i32 %65
  br label %BBExit

BBExit:                                      ; preds = %WhileExit3, %BBEntry5
  br label %WhileCmp6

WhileCmp6:                                   ; preds = %BBEntry8, %BBExit
  %66 = load i32, i32* %i2, align 4
  %67 = load i8*, i8** %str, align 8
  %68 = getelementptr i8, i8* %67, i32 %66
  %69 = load i8, i8* %68, align 1
  %70 = load i8, i8* @strend, align 1
  %71 = icmp ne i8 %69, %70
  %72 = load i32, i32* %i2, align 4
  %73 = load i8*, i8** %str, align 8
  %74 = getelementptr i8, i8* %73, i32 %72
  %75 = load i8, i8* %74, align 1
  %76 = load i8, i8* %seq, align 1
  %77 = icmp ne i8 %75, %76
  %78 = select i1 %71, i1 %77, i1 false
  br i1 %78, label %BBEntry8, label %WhileExit7
```

```
WhileExit7:                                       ; preds = %WhileCmp6
  %79 = load i32, i32* %i2, align 4
  store i32 %79, i32* %temp, align 4
  %80 = load i32, i32* %i2, align 4
  %81 = add i32 %80, 1
  store i32 %81, i32* %i2, align 4
  br label %WhileCmp

BBEntry8:                                         ; preds = %WhileCmp6
  %82 = load i32, i32* %i2, align 4
  %83 = add i32 %82, 1
  store i32 %83, i32* %i2, align 4
  br label %WhileCmp6
}

define i32 @charToInt(i8 %0) {
BBEntry:
  %chr = alloca i8, align 1
  store i8 %0, i8* %chr, align 1
  %1 = load i8, i8* %chr, align 1
  %2 = load i8, i8* @str0, align 1
  %3 = icmp eq i8 %1, %2
  br i1 %3, label %BBEntry1, label %BBExit

BBEntry1:                                         ; preds = %BBEntry
  ret i32 0
  br label %BBExit

BBExit:                                           ; preds = %BBEntry, %BBEntry1
  %4 = load i8, i8* %chr, align 1
  %5 = load i8, i8* @str1, align 1
  %6 = icmp eq i8 %4, %5
  br i1 %6, label %BBEntry2, label %BBExit3

BBEntry2:                                         ; preds = %BBExit
  ret i32 1
  br label %BBExit3
```

```
BBExit3:                                    ; preds = %BBExit, %BBEntry2
  %7 = load i8, i8* %chr, align 1
  %8 = load i8, i8* @str2, align 1
  %9 = icmp eq i8 %7, %8
  br i1 %9, label %BBEntry4, label %BBExit5

BBEntry4:                                   ; preds = %BBExit3
  ret i32 2
  br label %BBExit5

BBExit5:                                    ; preds = %BBExit3, %BBEntry4
  %10 = load i8, i8* %chr, align 1
  %11 = load i8, i8* @str3, align 1
  %12 = icmp eq i8 %10, %11
  br i1 %12, label %BBEntry6, label %BBExit7

BBEntry6:                                   ; preds = %BBExit5
  ret i32 3
  br label %BBExit7

BBExit7:                                    ; preds = %BBExit5, %BBEntry6
  %13 = load i8, i8* %chr, align 1
  %14 = load i8, i8* @str4, align 1
  %15 = icmp eq i8 %13, %14
  br i1 %15, label %BBEntry8, label %BBExit9

BBEntry8:                                   ; preds = %BBExit7
  ret i32 4
  br label %BBExit9

BBExit9:                                    ; preds = %BBExit7, %BBEntry8
  %16 = load i8, i8* %chr, align 1
  %17 = load i8, i8* @str5, align 1
  %18 = icmp eq i8 %16, %17
  br i1 %18, label %BBEntry10, label %BBExit11

BBEntry10:                                  ; preds = %BBExit9
  ret i32 5
  br label %BBExit11
```

```
BBExit11:                                   ; preds = %BBExit9, %BBEntry10
  ret i32 0
}

define i32 @gradeToInt(i8 %0) {
BBEntry:
  %chr = alloca i8, align 1
  store i8 %0, i8* %chr, align 1
  %1 = load i8, i8* %chr, align 1
  %2 = load i8, i8* @gradeA, align 1
  %3 = icmp eq i8 %1, %2
  br i1 %3, label %BBEntry1, label %BBExit

BBEntry1:                                   ; preds = %BBEntry
  ret i32 4
  br label %BBExit

BBExit:                                     ; preds = %BBEntry, %BBEntry1
  %4 = load i8, i8* %chr, align 1
  %5 = load i8, i8* @gradeB, align 1
  %6 = icmp eq i8 %4, %5
  br i1 %6, label %BBEntry2, label %BBExit3

BBEntry2:                                   ; preds = %BBExit
  ret i32 3
  br label %BBExit3

BBExit3:                                    ; preds = %BBExit, %BBEntry2
  %7 = load i8, i8* %chr, align 1
  %8 = load i8, i8* @gradeC, align 1
  %9 = icmp eq i8 %7, %8
  br i1 %9, label %BBEntry4, label %BBExit5

BBEntry4:                                   ; preds = %BBExit3
  ret i32 2
  br label %BBExit5

BBExit5:                                    ; preds = %BBExit3, %BBEntry4
```

```
  %10 = load i8, i8* %chr, align 1
  %11 = load i8, i8* @gradeD, align 1
  %12 = icmp eq i8 %10, %11
  br i1 %12, label %BBEntry6, label %BBExit7

BBEntry6:                                         ; preds = %BBExit5
  ret i32 1
  br label %BBExit7

BBExit7:                                          ; preds = %BBExit5, %BBEntry6
  %13 = load i8, i8* %chr, align 1
  %14 = load i8, i8* @gradeF, align 1
  %15 = icmp eq i8 %13, %14
  br i1 %15, label %BBEntry8, label %BBExit9

BBEntry8:                                         ; preds = %BBExit7
  ret i32 0
  br label %BBExit9

BBExit9:                                          ; preds = %BBExit7, %BBEntry8
  ret i32 0
}

define i32 @main() {
BBEntry:
  %coursename = alloca [6 x i8], align 1
  %coursescore = alloca i8, align 1
  store i32 0, i8* %coursescore, align 4
  %coursepre = alloca [1000 x i8], align 1
  %coursegrade = alloca i8, align 1
  store i32 0, i8* %coursegrade, align 4
  %oneline = alloca [1000 x i8], align 1
  %firstchar = alloca i8, align 1
  store i32 0, i8* %firstchar, align 4
  %tempchar = alloca i8, align 1
  store i32 0, i8* %tempchar, align 4
  %score = alloca i32, align 4
  store i32 0, i32* %score, align 4
  %grade = alloca i32, align 4
  store i32 0, i32* %grade, align 4
  %score_learn = alloca i32, align 4
  store i32 0, i32* %score_learn, align 4
  %score_get = alloca i32, align 4
  store i32 0, i32* %score_get, align 4
  %score_remain = alloca i32, align 4
  store i32 0, i32* %score_remain, align 4
  %gpa = alloca i32, align 4
  store i32 0, i32* %gpa, align 4
  %cnt = alloca i32, align 4
  store i32 0, i32* %cnt, align 4
  %names = alloca [1000 x i8], align 1
  %pres = alloca [1000000 x i8], align 1
  %learned = alloca [100 x i32], align 4
  %learnable = alloca [100 x i32], align 4
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  %j = alloca i32, align 4
  store i32 0, i32* %j, align 4
  %index = alloca i32, align 4
  store i32 0, i32* %index, align 4
  %cursor = alloca i32, align 4
  store i32 0, i32* %cursor, align 4
  %orCnt = alloca i32, align 4
  store i32 0, i32* %orCnt, align 4
  %andCnt = alloca i32, align 4
  store i32 0, i32* %andCnt, align 4
  %learn = alloca i32, align 4
  store i32 0, i32* %learn, align 4
  %orStr = alloca [100 x i8], align 1
  %andStr = alloca [6 x i8], align 1
  br label %WhileCmp

WhileCmp:                                         ; preds = %BBExit39, %BBEntry
  %0 = call i32 (i8*, ...) @scank(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @0, i32 0, i32 0), i8* %firstchar)
  %1 = load i8, i8* %firstchar, align 1
  %2 = load i8, i8* @strnewline, align 1
  %3 = icmp eq i8 %1, %2
  br i1 %3, label %BBEntry2, label %BBExit33
```

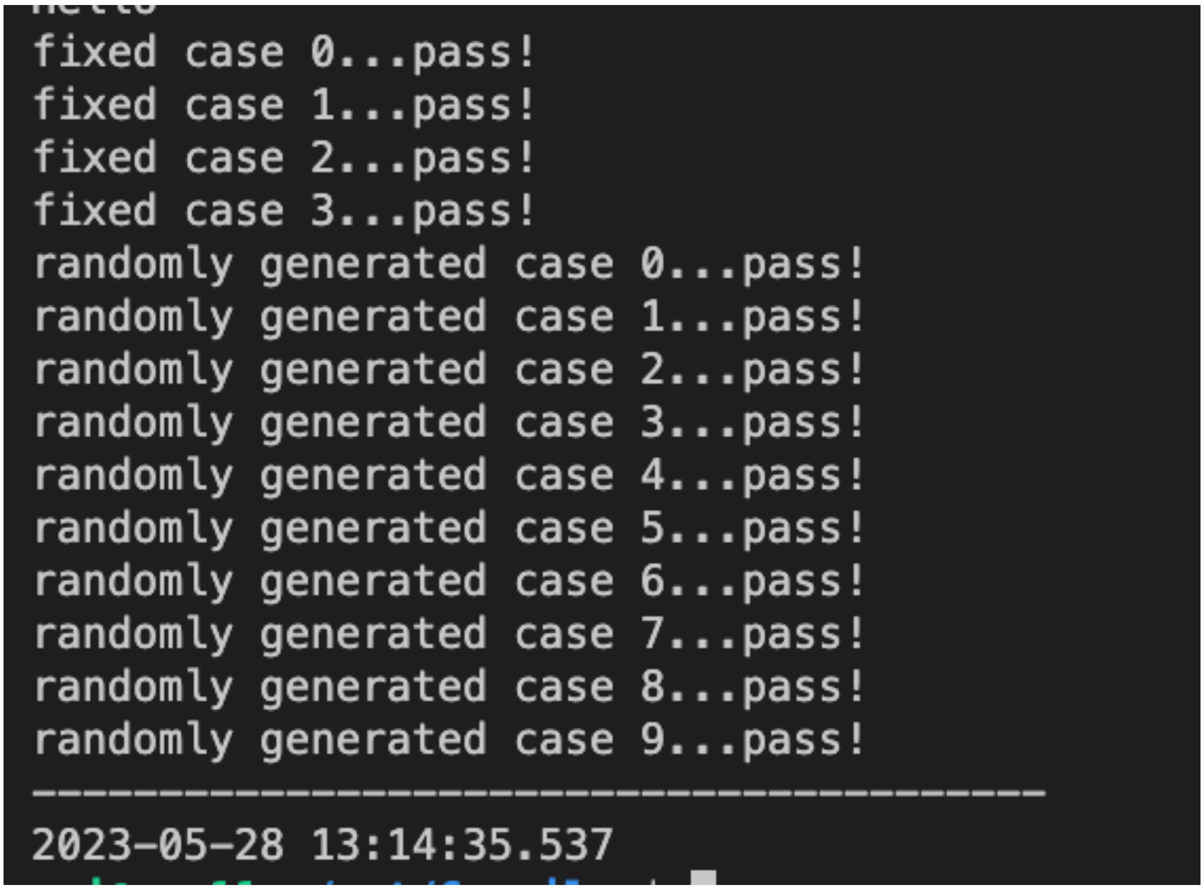
```
BBEntry2:                                          ; preds = %WhileCmp
  %4 = bitcast [1000 x i8]* %names to i8*
  %5 = call i32 @charLeaveStr(i8* %4)
  %6 = bitcast [1000000 x i8]* %pres to i8*
  %7 = call i32 @charLeaveStr(i8* %6)
  store i32 0, i32* %cursor, align 4
  br label %ForCmp

ForCmp:                                            ; preds = %BBExit20, %BBEntry2
  %8 = load i32, i32* %cursor, align 4
  %9 = load i32, i32* %cnt, align 4
  %10 = icmp slt i32 %8, %9
  br i1 %10, label %BBEntry3, label %ForExit

ForExit:                                           ; preds = %ForCmp
  %11 = load i32, i32* %score_learn, align 4
  %12 = icmp ne i32 %11, 0
  br i1 %12, label %BBEntry21, label %BBEntry22

BBEntry3:                                          ; preds = %ForCmp
  %13 = load i32, i32* %cursor, align 4
  %14 = bitcast [100 x i32]* %learned to i32*
  %15 = getelementptr i32, i32* %14, i32 %13
  %16 = load i32, i32* %15, align 4
  br i32 %16, label %BBEntry4, label %BBEntry5

BBEntry4:                                          ; preds = %BBEntry3
  %17 = load i32, i32* %cursor, align 4
  %18 = bitcast [100 x i32]* %learnable to i32*
  %19 = getelementptr i32, i32* %18, i32 %17
  store i32 0, i32* %19, align 4
  br label %BBExit20

BBEntry5:                                          ; preds = %BBEntry3
  %20 = load i32, i32* %cursor, align 4
  %21 = bitcast [100 x i32]* %learnable to i32*
  %22 = getelementptr i32, i32* %21, i32 %20
  store i32 0, i32* %22, align 4
  %23 = bitcast [1000 x i8]* %coursepre to i8*
  %24 = bitcast [1000000 x i8]* %pres to i8*
  %25 = load i32, i32* %cursor, align 4
  %26 = load i8, i8* @strspace, align 1
  %27 = call i32 @getSubStr(i8* %23, i8* %24, i32 %25, i8 %26)
  %28 = bitcast [1000 x i8]* %coursepre to i8*
  %29 = load i8, i8* @strsemicolon, align 1
  %30 = call i32 @getSubCnt(i8* %28, i8 %29)
  store i32 %30, i32* %orCnt, align 4
  %31 = load i32, i32* %orCnt, align 4
  %32 = icmp eq i32 %31, 0
  br i1 %32, label %BBEntry6, label %BBExit

BBEntry6:                                    ; preds = %BBEntry5
  %33 = load i32, i32* %cursor, align 4
  %34 = bitcast [100 x i32]* %learnable to i32*
  %35 = getelementptr i32, i32* %34, i32 %33
  store i32 1, i32* %35, align 4
  br label %BBExit

BBExit:                                      ; preds = %BBEntry5, %BBEntry6
  store i32 0, i32* %i, align 4
  br label %ForCmp7

ForCmp7:                                     ; preds = %ForExit13, %BBExit
  %36 = load i32, i32* %i, align 4
  %37 = load i32, i32* %orCnt, align 4
  %38 = icmp slt i32 %36, %37
  %39 = load i32, i32* %cursor, align 4
  %40 = bitcast [100 x i32]* %learnable to i32*
  %41 = getelementptr i32, i32* %40, i32 %39
  %42 = load i32, i32* %41, align 4
  %43 = icmp eq i32 %42, 0
  %44 = select i1 %38, i1 %43, i1 false
  br i1 %44, label %BBEntry10, label %BBExit20

BBEntry10:                                   ; preds = %ForCmp7
  store i32 1, i32* %learn, align 4
  %45 = bitcast [100 x i8]* %orStr to i8*
  %46 = bitcast [1000 x i8]* %coursepre to i8*
  %47 = load i32, i32* %i, align 4
  %48 = load i8, i8* @strsemicolon, align 1
  %49 = call i32 @getSubStr(i8* %45, i8* %46, i32 %47, i8 %48)
  %50 = bitcast [100 x i8]* %orStr to i8*
  %51 = load i8, i8* @strcomma, align 1
  %52 = call i32 @getSubCnt(i8* %50, i8 %51)
  store i32 %52, i32* %andCnt, align 4
  store i32 0, i32* %j, align 4
  br label %ForCmp11

ForCmp11:                                   ; preds = %BBExit19, %BBEntry10
  %53 = load i32, i32* %j, align 4
  %54 = load i32, i32* %andCnt, align 4
  %55 = icmp slt i32 %53, %54
  %56 = load i32, i32* %learn, align 4
  %57 = icmp eq i32 %56, 1
  %58 = select i1 %55, i1 %57, i1 false
  br i1 %58, label %BBEntry14, label %ForExit13

ForExit13:                                  ; preds = %ForCmp11
  %59 = load i32, i32* %learn, align 4
  %60 = load i32, i32* %cursor, align 4
  %61 = bitcast [100 x i32]* %learnable to i32*
  %62 = getelementptr i32, i32* %61, i32 %60
  store i32 %59, i32* %62, align 4
  %63 = load i32, i32* %i, align 4
  %64 = add i32 %63, 1
  store i32 %64, i32* %i, align 4
  br label %ForCmp7

BBEntry14:                                  ; preds = %ForCmp11
  %65 = bitcast [6 x i8]* %andStr to i8*
  %66 = bitcast [100 x i8]* %orStr to i8*
  %67 = load i32, i32* %j, align 4
  %68 = load i8, i8* @strcomma, align 1
  %69 = call i32 @getSubStr(i8* %65, i8* %66, i32 %67, i8 %68)
  %70 = bitcast [6 x i8]* %andStr to i8*
  %71 = bitcast [1000 x i8]* %names to i8*
  %72 = load i8, i8* @strspace, align 1
  %73 = call i32 @getStrIndex(i8* %70, i8* %71, i8 %72)
  store i32 %73, i32* %index, align 4
  %74 = load i32, i32* %index, align 4
  %75 = icmp eq i32 %74, 0
  br i1 %75, label %BBEntry15, label %BBEntry16

BBEntry15:                                        ; preds = %BBEntry14
  store i32 0, i32* %learn, align 4
  br label %BBExit19

BBEntry16:                                        ; preds = %BBEntry14
  %76 = load i32, i32* %index, align 4
  %77 = sub i32 %76, 1
  store i32 %77, i32* %index, align 4
  %78 = load i32, i32* %index, align 4
  %79 = bitcast [100 x i32]* %learned to i32*
  %80 = getelementptr i32, i32* %79, i32 %78
  %81 = load i32, i32* %80, align 4
  %82 = icmp eq i32 %81, 0
  br i1 %82, label %BBEntry17, label %BBExit19

BBEntry17:                                        ; preds = %BBEntry16
  store i32 0, i32* %learn, align 4
  br label %BBExit19

BBExit19:                                         ; preds = %BBEntry17, %BBEntry16, %BBEntry15
  %83 = load i32, i32* %j, align 4
  %84 = add i32 %83, 1
  store i32 %84, i32* %j, align 4
  br label %ForCmp11

BBExit20:                                         ; preds = %ForCmp7, %BBEntry4
  %85 = load i32, i32* %cursor, align 4
  %86 = add i32 %85, 1
  store i32 %86, i32* %cursor, align 4
  br label %ForCmp

BBEntry21:                                        ; preds = %ForExit
```

```
  %87 = load i32, i32* %gpa, align 4
  %88 = sitofp i32 %87 to double
  %89 = fmul double 1.000000e+00, %88
  %90 = load i32, i32* %score_learn, align 4
  %91 = sitofp i32 %90 to double
  %92 = fdiv double %89, %91
  %93 = call i32 (i32*, ...) @printk(i32* bitcast ([11 x i8]* @1 to i32*), double %92)
  br label %BBExit23

BBEntry22:                                        ; preds = %ForExit
  %94 = call i32 (i32*, ...) @printk(i32* bitcast ([10 x i8]* @2 to i32*))
  br label %BBExit23

BBExit23:                                         ; preds = %BBEntry22, %BBEntry21
  %95 = load i32, i32* %score_learn, align 4
  %96 = call i32 (i32*, ...) @printk(i32* bitcast ([21 x i8]* @3 to i32*), i32 %95)
  %97 = load i32, i32* %score_get, align 4
  %98 = call i32 (i32*, ...) @printk(i32* bitcast ([21 x i8]* @4 to i32*), i32 %97)
  %99 = load i32, i32* %score_remain, align 4
  %100 = call i32 (i32*, ...) @printk(i32* bitcast ([24 x i8]* @5 to i32*), i32 %99)
  %101 = call i32 (i32*, ...) @printk(i32* bitcast ([31 x i8]* @6 to i32*))
  %102 = load i32, i32* %score_remain, align 4
  %103 = icmp ne i32 %102, 0
  br i1 %103, label %BBEntry24, label %BBEntry31

BBEntry24:                                        ; preds = %BBExit23
  store i32 0, i32* %cursor, align 4
  br label %ForCmp25

ForCmp25:                                         ; preds = %BBExit30, %BBEntry24
  %104 = load i32, i32* %cursor, align 4
  %105 = load i32, i32* %cnt, align 4
  %106 = icmp slt i32 %104, %105
  br i1 %106, label %BBEntry28, label %BBExit32

BBEntry28:                                        ; preds = %ForCmp25
  %107 = load i32, i32* %cursor, align 4
  %108 = bitcast [100 x i32]* %learnable to i32*
  %109 = getelementptr i32, i32* %108, i32 %107
  %110 = load i32, i32* %109, align 4
  %111 = icmp eq i32 %110, 1
  br i1 %111, label %BBEntry29, label %BBExit30

BBEntry29:                                        ; preds = %BBEntry28
  %112 = bitcast [6 x i8]* %coursename to i8*
  %113 = bitcast [1000 x i8]* %names to i8*
  %114 = load i32, i32* %cursor, align 4
  %115 = load i8, i8* @strspace, align 1
  %116 = call i32 @getSubStr(i8* %112, i8* %113, i32 %114, i8 %115)
  %117 = call i32 (i32*, ...) @printk(i32* bitcast ([6 x i8]* @7 to i32*), [6 x i8]* %coursename)
  br label %BBExit30

BBExit30:                                         ; preds = %BBEntry28, %BBEntry29
  %118 = load i32, i32* %cursor, align 4
  %119 = add i32 %118, 1
  store i32 %119, i32* %cursor, align 4
  br label %ForCmp25

BBEntry31:                                        ; preds = %BBExit23
  %120 = call i32 (i32*, ...) @printk(i32* bitcast ([27 x i8]* @8 to i32*))
  br label %BBExit32

BBExit32:                                         ; preds = %ForCmp25, %BBEntry31
  ret i32 0
  br label %BBExit33

BBExit33:                                         ; preds = %WhileCmp, %BBExit32
  %121 = call i32 (i8*, ...) @scank(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @9, i32 0, i32 0), [1000 x i8]*
%oneline)
  %122 = call i32 (i8*, ...) @scank(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @10, i32 0, i32 0), i8*
%tempchar)
  %123 = load i8, i8* %firstchar, align 1
  %124 = bitcast [1000 x i8]* %oneline to i8*
  %125 = call i32 @charAddStr(i8 %123, i8* %124)
  store i32 0, i32* %cursor, align 4
  %126 = bitcast [6 x i8]* %coursename to i8*
  %127 = bitcast [1000 x i8]* %oneline to i8*
  %128 = load i32, i32* %cursor, align 4
```

```
%129 = load i8, i8* @strseq, align 1
%130 = call i32 @copyStr(i8* %126, i8* %127, i32 %128, i8 %129)
%131 = bitcast [6 x i8]* %coursename to i8*
%132 = call i32 @myStrlen(i8* %131)
%133 = add i32 %132, 1
store i32 %133, i32* %cursor, align 4
%134 = load i32, i32* %cursor, align 4
%135 = bitcast [1000 x i8]* %oneline to i8*
%136 = getelementptr i8, i8* %135, i32 %134
%137 = load i8, i8* %136, align 1
store i8 %137, i8* %coursescore, align 1
%138 = load i32, i32* %cursor, align 4
%139 = add i32 %138, 2
store i32 %139, i32* %cursor, align 4
%140 = bitcast [1000 x i8]* %coursepre to i8*
%141 = bitcast [1000 x i8]* %oneline to i8*
%142 = load i32, i32* %cursor, align 4
%143 = load i8, i8* @strseq, align 1
%144 = call i32 @copyStr(i8* %140, i8* %141, i32 %142, i8 %143)
%145 = load i32, i32* %cursor, align 4
%146 = bitcast [1000 x i8]* %coursepre to i8*
%147 = call i32 @myStrlen(i8* %146)
%148 = add i32 %145, %147
%149 = add i32 %148, 1
store i32 %149, i32* %cursor, align 4
%150 = load i32, i32* %cursor, align 4
%151 = bitcast [1000 x i8]* %oneline to i8*
%152 = getelementptr i8, i8* %151, i32 %150
%153 = load i8, i8* %152, align 1
store i8 %153, i8* %coursegrade, align 1
%154 = load i8, i8* %coursescore, align 1
%155 = call i32 @charToInt(i8 %154)
store i32 %155, i32* %score, align 4
%156 = load i8, i8* %coursegrade, align 1
%157 = call i32 @gradeToInt(i8 %156)
store i32 %157, i32* %grade, align 4
%158 = load i8, i8* %coursegrade, align 1
%159 = load i8, i8* @gradeF, align 1
%160 = icmp eq i8 %158, %159
```

```
  br i1 %160, label %BBEntry34, label %BBEntry35

BBEntry34:                                        ; preds = %BBExit33
  %161 = load i32, i32* %score_learn, align 4
  %162 = load i32, i32* %score, align 4
  %163 = add i32 %161, %162
  store i32 %163, i32* %score_learn, align 4
  %164 = load i32, i32* %score_remain, align 4
  %165 = load i32, i32* %score, align 4
  %166 = add i32 %164, %165
  store i32 %166, i32* %score_remain, align 4
  %167 = load i32, i32* %cnt, align 4
  %168 = bitcast [100 x i32]* %learned to i32*
  %169 = getelementptr i32, i32* %168, i32 %167
  store i32 0, i32* %169, align 4
  br label %BBExit39

BBEntry35:                                        ; preds = %BBExit33
  %170 = load i8, i8* %coursegrade, align 1
  %171 = load i8, i8* @strend, align 1
  %172 = icmp eq i8 %170, %171
  br i1 %172, label %BBEntry36, label %BBEntry37

BBEntry36:                                        ; preds = %BBEntry35
  %173 = load i32, i32* %score_remain, align 4
  %174 = load i32, i32* %score, align 4
  %175 = add i32 %173, %174
  store i32 %175, i32* %score_remain, align 4
  %176 = load i32, i32* %cnt, align 4
  %177 = bitcast [100 x i32]* %learned to i32*
  %178 = getelementptr i32, i32* %177, i32 %176
  store i32 0, i32* %178, align 4
  br label %BBExit39

BBEntry37:                                        ; preds = %BBEntry35
  %179 = load i32, i32* %score_learn, align 4
  %180 = load i32, i32* %score, align 4
  %181 = add i32 %179, %180
  store i32 %181, i32* %score_learn, align 4
  %182 = load i32, i32* %score_get, align 4
  %183 = load i32, i32* %score, align 4
  %184 = add i32 %182, %183
  store i32 %184, i32* %score_get, align 4
  %185 = load i32, i32* %gpa, align 4
  %186 = load i32, i32* %grade, align 4
  %187 = load i32, i32* %score, align 4
  %188 = mul i32 %186, %187
  %189 = add i32 %185, %188
  store i32 %189, i32* %gpa, align 4
  %190 = load i32, i32* %cnt, align 4
  %191 = bitcast [100 x i32]* %learned to i32*
  %192 = getelementptr i32, i32* %191, i32 %190
  store i32 1, i32* %192, align 4
  br label %BBExit39

BBExit39:                                         ; preds = %BBEntry36, %BBEntry37, %BBEntry34
  %193 = load i32, i32* %cnt, align 4
  %194 = add i32 %193, 1
  store i32 %194, i32* %cnt, align 4
  %195 = bitcast [1000 x i8]* %names to i8*
  %196 = bitcast [6 x i8]* %coursename to i8*
  %197 = load i8, i8* @strspace, align 1
  %198 = call i32 @appendStr(i8* %195, i8* %196, i8 %197)
  %199 = bitcast [1000000 x i8]* %pres to i8*
  %200 = bitcast [1000 x i8]* %coursepre to i8*
  %201 = load i8, i8* @strspace, align 1
  %202 = call i32 @appendStr(i8* %199, i8* %200, i8 %201)
  br label %WhileCmp
}
```

运行截图



```
fixed case 0...pass!
fixed case 1...pass!
fixed case 2...pass!
fixed case 3...pass!
randomly generated case 0...pass!
randomly generated case 1...pass!
randomly generated case 2...pass!
randomly generated case 3...pass!
randomly generated case 4...pass!
randomly generated case 5...pass!
randomly generated case 6...pass!
randomly generated case 7...pass!
randomly generated case 8...pass!
randomly generated case 9...pass!
------------------------------------------
2023-05-28 13:14:35.537
```

附：提交说明

本次提交主要使用学在浙大进行提交，同时为了避免这个系统出现常见的无法解释的问题(赶在DDL提交 是会出现各种事故的，助教不会视作不可抗力，请同学们自行斟酌)，助教也同时接受钉钉/邮箱提交(这 两个工具都有时间戳)。

提交DDL: 2023.5.28 22:00 一旦延后评分下降半个评级，然后每隔12小时再下降半个评级。例如:假 如你本可以评分为A+，但是你不小心在2023.5.28 22:01:00 才提交代码，于是你的评分可能降为A，如 果你把“十点”记成了2023.5.29 上午十点,却又不小心在2023.5.29 10:01:00才提交代码，那你的评分确 定从A+ -> A了。

提交的内容一定是一个压缩包，压缩包命名为组名**_2023编译原理大程，**压缩包内有包含组员的学号和 名字的文件以及实验报告以及源代码**。

提交时在学在浙大上一般由组长提交一份，但是为了避免组长出状况，组员也可以自己提交。