一、内容

一）目标

设计并实现 Lexical Analyzer 中的 C 语言子集的目标代码生成程序，并打印分析结果。实现以下功能：

1、

a) 能够生成汇编语言形式的目标代码，应至少包括以下代码类型：赋值语句、算术运算操作（加减乘除），跳转语句、分支与循环语句及其他基本语句。

b) 目标代码能够在 Mars 上运行。

2、

a) 生成过程/函数的目标代码

二）C 语言子集：

数据类型: **int**, 无符号整数, 取值范围 0-9999

```
int a;
int a,b;
int a = 1;
```

算术运算符: **+,-**

```
a = b + 1;
a = b + c;
```

赋值运算符: **=**

```
a = 1;
```

关系运算符: **== ,>,<,!=,>=,<=**

```
a = (b==c);
a = (b>c);
a = (b<c);
```

逻辑运算符: **&&, ||, !**

```
a = (b&&c);
a = (b||c);
a = (!b);
```

条件语句: **if**

```
if(a==b)
{
};
```

循环语句: **while**

```
while(a==b)
{

};
```

输入,输出: **get**,**put**

get(a);

put(a);

语句结束符: **;**

条件语句 **if else**

if(a==b)

{

};

else

{

};

三）目标语言

假设数据类型只包含整数类型，不包含如浮点数、数组、结构和指针等其它数据类型， 目标语言选定为 MIPS32 指令序列，中间代码及 MIPS32 指令对应关系如下表所示。其中 reg(x)表示变量 x 所分配的寄存器。

中间代码	MIPS32 指令
LABEL x	x：
x :=#k	li　reg(x),k
x := y	move　reg(x), reg(y)
x := y + z	add　reg(x), reg(y) , reg(z)
x := y - z	sub　reg(x), reg(y) , reg(z)
x := y * z	mul　reg(x), reg(y) , reg(z)
x := y / z	div reg(y) , reg(z) mflo reg(x)
GOTO x	j　x
RETURN x	move $v0, reg(x) jr $ra
IF x==y GOTO z	beq reg(x),reg(y),z
IF x!=y GOTO z	bne reg(x),reg(y),z
IF x>y GOTO z	bgt reg(x),reg(y),z
IF x>=y GOTO z	bge reg(x),reg(y),z
IF x<y GOTO z	ble reg(x),reg(y),z
IF x<=y GOTO z	blt reg(x),reg(y),z
X:=CALL f	jal f move reg(x),$v0

二、过程或算法（源程序）

1. 目标代码生成程序总体说明：

实现了目标代码的生成并能在 Mars 上运行得到运行结果，完成了一个能够生成可以在 Mars 运行的汇编码的 C 语言子集编译器。采用将源程序直接翻译为目标代码的方式。做了一些简化处理，假设数据类型只包含整数类型，不包含如浮点数、数组、结构和指针等其它数据类型，目标语言为汇编语言。

可以为赋值语句、算术运算操作（加减乘除），跳转语句、分支（if-else）与循环语句（while）及其他基本语句（逻辑运算操作等）生成汇编代码。

将目标语言选定为 MIPS32 指令序列。为简便起见，将所有结果，包括中间结果均存在内存中，这样可以解决寄存器数量不够的问题。

本程序的代码在 Code 目录下。共包含 7 个文件。grammars.txt 中是适应于程序输入接口的文法，该文件非程序运行所必需，运行时，只需将该文件中内容全部复制到命令行界面中并输入 end+回车即可。inputs.c 文件中是待编译的源代码，要替换测试样例，只需替换该文件中的内容即可。lexical_analyzer.h 是词法分析器头文件，lexical_tool.h 中是词法分析中用到的工具函数。obj_code.asm 中是生成的汇编码，可将该文件中的汇编码全部复制到 Mars 中运行。该文件非运行程序所必需，运行程序后会自动生成。syntax_analyzer.cpp 文件中的代码同时实现了语法分析、语义分析、目标代码生成以及相关报错功能，该文件也是程序的主文件，要运行程序，需编译运行该文件。Tree.txt 是程序运行生成的语法分析树，非程序运行所必需，可将该文件内容复制到网站 http://www.webgraphviz.com/绘制语法分析树。

2. 存储分配方法说明：

将所有结果，包括中间结果均存在内存中。在声明时就为每个标识符分配内存地址，也为每个数字常量分配内存地址，这样就可以统一处理数字常量和标识符了，只需要访问它们的内存地址即可。使用 get 函数读入变量值时，直接将其存放在对应标识符的内存地址处。在有必要时，为语法中的非终结符也分配内存地址，用于存放中间变量。这样做的好处是可以解决寄存器数量不够的问题。

在这种方法中，实际用来长期存储数据的容器是内存，寄存器仅在进行运算时临时存储运算过程中的值，或进行参数传递使用。

具体的内存分配方法，则是在每次需要一个新的内存地址时，线性开辟下一个尚未使用的内存地址。在程序中用一个 vector 模拟内存空间，因为内存地址按字节对齐，一个机器字占 4 个字节，且栈地址从高地址端向低地址端增长，因此 vector 下标乘以负四，就是对应内存中的虚拟地址。

3. 错误处理

可以检测 get 函数接收的参数是否为左值，在不是左值时报错。具体做法是，在参数列表中的非终结符 expr 展开过程中，必须每一步展开时产生式体中除最左边

的非终结符外的所有非终结符均推出空，才能保证该 expr 只产生一个标识符而非表达式。观察到当某一步不满足上述条件时，其后面的非终结符地址会发生变化，可通过判断语义动作 6 前面一个非终结符的 saddr 和 iaddr 是否相等，来判断该非终结符是否为空。只有当满足上述条件时，才把语义动作 5 前的非终结符的 sname 传给父结点，而只有当最底端推出标识符时，才把标识符的名字往上传递。因此只有当 expr 产生一个终结符时，expr 的 sname 才为 id 的名字，否则为非终结符的名字，可通过 expr 的 sname 是否出现在符号表中来判断其是否为左值。

此外还涵盖了下述错误检查功能。

对整型（int）及布尔变量的类型检查，两类变量不能相互赋值及运算；仅整型变量才能参与算术运算；能够检查出未声明的变量。能够处理“丢失右括号”，“丢失运算符”和“丢失操作数”等报错信息。

能判断源代码是否符合以下语义假设并给出相应错误具体位置：

过程/函数仅能定义一次、程序中所有变量均不能重名、过程/函数不可嵌套定义；能够定位源代码中的错误位置。

4. 程序说明：

符号表的结构与 Syntax-DirectedTranslation 中一致。

符号表结构如下：

字段	类型	说明
name	string	标识符的名字
type	string	标识符的类型，缺省值为“unknown”
p	pos	标识符在源代码中出现的位置（行，列）
val	int	标识符的值（如果有）
addr	string	存储标识符值的地址

对 Syntax-DirectedTranslation 的语法翻译制导方案进行修改，使之能够生成目标代码。

修改后的语法翻译方案如下：

$$S \rightarrow T\ S \mid O\ S \mid C\ S \mid W\ S \mid id\ 11\ CC\ S \mid ST\ S \mid ;\ S \mid \varepsilon$$
$$11 \rightarrow \varepsilon$$

$T \rightarrow int\ decs\ ;\ |\ \_Bool\ decs\ ;$
$decs \rightarrow dec\ dec'$
$dec' \rightarrow ,decs \mid \varepsilon$
$dec \rightarrow id\ 1\ assign$
$assign \rightarrow \varepsilon \mid\ =\ expr\ 12$
$O \rightarrow (id\ ) = expr;$
$1 \rightarrow \varepsilon$
$expr \rightarrow G\ 5\ expr'\ 6$
$expr' \rightarrow\ \&\&\ G\ 8\ expr'\ 9 \mid \backslash|\backslash|\ G\ 8\ expr'\ 9 \mid \varepsilon\ 10$
$G \rightarrow\ L\ 5\ G'\ 6$
$G' \rightarrow \backslash|\ L\ 8\ G'\ 9 \mid \varepsilon\ 10$
$L \rightarrow M\ 5\ L'\ 6$
$L' \rightarrow \&\ M\ 8\ L'\ 9 \mid \varepsilon\ 10$
$M \rightarrow H\ 5\ M'\ 6$
$M' \rightarrow == H\ 8\ M'\ 9 \mid\ !=H\ 8\ M'\ 9 \mid \varepsilon\ 10$
$H \rightarrow I\ 5\ H'\ 6$
$H' \rightarrow >\ I\ 8\ H'\ 9 \mid < I\ 8\ H'\ 9 \mid\ >= I\ 8\ H'\ 9 \mid <= I\ 8\ H'\ 9 \mid \varepsilon\ 10$
$I \rightarrow J\ 5\ I'\ 6$
$I' \rightarrow\ +J\ 8\ I'\ 9 \mid -J\ 8\ I'\ 9 \mid \varepsilon\ 10$
$J \rightarrow K\ 5\ J'\ 6$
$J' \rightarrow *\ K\ 8\ J'\ 9 \mid\ /\ K\ 8\ J'\ 9 \mid \varepsilon\ 10$
$5 \rightarrow\ \varepsilon$
$6 \rightarrow\ \varepsilon$
$8 \rightarrow\ \varepsilon$
$9 \rightarrow\ \varepsilon$
$7 \rightarrow\ \varepsilon$
$10 \rightarrow\ \varepsilon$
$K \rightarrow\ !\ K\ 4 \mid (expr)4 \mid id\ 3 \mid decimal\ 7 \mid hex\ 4$
$3 \rightarrow\ \varepsilon$
$C \rightarrow if\ (\ expr\ )\ 17\ \{\ S\ 18\ \}\ ;\ 19\ EL\ 20$
$EL \rightarrow \varepsilon \mid else\ \{\ S\ \};$
$W \rightarrow while\ (\ 24\ expr\ )\ 21\ \{\ S\ 22\ \}\ 23\ ;$
$CC \rightarrow (\ 14\ expr\_list) \mid\ = expr\ 13\ ;$
$12 \rightarrow \varepsilon$
$13 \rightarrow \varepsilon$
$4 \rightarrow\ \varepsilon$
$14 \rightarrow \varepsilon$
$expr\_list \rightarrow expr\ 16\ E$
$E \rightarrow ,\ expr\ 15\ E \mid \varepsilon$
$15 \rightarrow \varepsilon$
$16 \rightarrow \varepsilon$
$17 \rightarrow \varepsilon$
$18 \rightarrow \varepsilon$
$19 \rightarrow \varepsilon$

$20 \rightarrow \varepsilon$
$21 \rightarrow \varepsilon$
$22 \rightarrow \varepsilon$
$23 \rightarrow \varepsilon$
$24 \rightarrow \varepsilon$
$ST \rightarrow \ struct\ id\ 2\ \{\ ST'\ \}\ ;$
$2 \rightarrow \ \backslash e$
$ST' \rightarrow T\ ST' \mid ST\ ST' \mid ;ST' \mid \varepsilon$

文法中标号与语义动作的对应表如下：

标号	语义动作

1	{if(symtable.count(node(1).sname))raiseError else
	symtable[node(1).sname] = symt_attr(node(1).stype,
	node(1).sp,id_val.size()); node(0).stype=node(1).stype;
	node(3).iname = node(1).sname;
	node(3).itype = node(0).stype;
	node(1).saddr = new addr;
	symtable[node(1).sname].addr = node(1).saddr;}

2	{ if(symtable.count(node(2).sname))raiseError else symtable[node(2).sname] = symt_attr(node(2).stype, node(2).sp); node(2).stype=”struct” }

3	{if(symtable.count(node(1).sname))raiseError else begin
	node(1).stype = symtable[node(1).sname].stype node(1).saddr
	= symtable[node(1).sname].saddr end;
	node(0).stype = node(1).stype;
	node(0).sp = node(1).sp;
	node(0).saddr = node(1).saddr;
	node(0).sname = node(1).sname; }

4	{ node(0).sp = node(1).sp}
5	{node(3).itype = node(1).stype; node(3).iaddr =node(1).saddr; node(3).ip = node(1).sp }

6	{node(0).stype = node(3).stype;
	node(0).saddr= node(3).saddr;
	node(0).sp = node(1).sp;
	if(node(3).saddr==node(3).iaddr) node(0).sname =
	node(1).sname;}

7	{node(0).sval = int(node(1).sval);
	node(0).sp = node(1).sp;
	node(0).saddr = node(1).saddr;
	genCode(li $t0, node(0).sval; sw $t0, -4*node(1).saddr($sp) }

8	{if(node(0).itype != node(2).stype)raiseError;
	node(4).itype = operator(1).type;
	node(4).iaddr = new addr;

	id_val[node(4).iaddr] = id_val[node(0).iaddr] operator(1) id_val[node(2).saddr]; genCode(id_val[node(4).iaddr] = id_val[node(0).iaddr] operator(1) id_val[node(2).saddr])}
9	{node(0).stype = node(4).stype; node(0).saddr = node(4).saddr;}
10	{node(0).stype = node(0).itype; node(0).saddr = node(0).iaddr;}
11	{node(3).iname = node(1).sname; node(3).ip = node(1).sp}
12	{if(symtable[node(0).iname].type!=node(2).stype)raiseError; id_val[symtable[node(0).iname].addr]=id_val[node(2).saddr]; genCode(id_val[symtable[node(0).iname].addr]=id_val[node(2).saddr])}

13	{ if(symtable.count(node(0).iname))rasieError else begin
	if(symtable[node(0).iname].type!=node(2).stype)raiseError;
	id_val[symtable[node(0).iname].addr]=id_val[node(2).saddr];
	genCode(id_val[symtable[node(0).iname].addr]=id_val[node(2).saddr]) end}

14	{node(3).iname = node(0).iname;}
15	{if(!symtable.count(node(2).sname)) raiseError else get(node(2).sname) }
16	{node(3).iname = node(0).iname; if(node(0).iname==”get”) if(!symtable.count(node(1).sname)) raiseError else get(symtable[node(1).sname].addr) else if(node(0).iname==”put”)put(symtable[node(1).sname].addr) }
17	{label1 = new label; genCode(beqz node(3).saddr, label1); node(11).ilabel = label1;}
18	{label2 = new label; genCode(j label2)}
19	{attach label1}
20	{attach label2}
21	{label2 = new label;

```
        genCode(beqz node(4).saddr, label2);
        node(11).ilabel = label}
22      {genCode(j label1)}
23      {attach label2}
24      { label1 = new label;
        attach label1;
        node(3).ilabel=label3}
```

首先编写两个工具函数，用于根据给定寄存器和地址生成对应的 sw 和 lw 汇编码。

```
239 void sw(string& reg,int addr){
240     /*
241     save word to memory from reg according to addr
242     */
243     addr*=-4;
244     FILE* fout = fopen(obj_fn,"a");
245     fprintf(fout,(string("    sw ")+reg + string(", ")+to_string(addr)+
246         string("($sp)\n")).c_str());
247     fclose(fout);
248 }
```

```
358 void lw(string& reg,int addr){
359     /*
360     load word
361     */
362     addr*=-4;
363     FILE* fout = fopen(obj_fn,"a");
364     fprintf(fout,(string("    lw ")+reg + string(", ")+to_string(addr)+
365     string("($sp)\n")).c_str());
366     fclose(fout);
367 }
```

对于输入输出函数 get 和 put，在文法中将函数名以继承属性的形式向下传递，传递给后面的 expr_list 和 E 等非终结符。在这些非终结符展开时，对函数名进行判断。下图中为读写函数的判断逻辑实现。

```
618 case 16:
619     gptree[pare->neibs[2]].iname = pare->iname;
620     if(pare->iname == "get"){
621         if(!symtable.count(gptree[pare->neibs[0]].sname)){
622             printf("%s:%d:%d: error: expression must be an lvalue\n",
623             fn,gptree[pare->neibs[0]].sp.line,
624             gptree[pare->neibs[0]].sp.column);
625         }else{
626             readid(gptree[pare->neibs[0]].saddr);
627         }
628     }else if(pare->iname == "put"){
629         writeid(gptree[pare->neibs[0]].saddr);
630     }
```

若函数名为 get，还需判断参数列表中的参数是否为左值，若为左值，调用读函数生成将用户输入读入到标识符所在地址的汇编码。下图中为读函数实现。

```
248 void readid(int addr){
249     /*
250     read identifier in asm code with given identifier name
251     */
252     readop();
253     string reg = "$v0";
254     sw(reg,addr);
255 }
```

```
226 void readop(){
227     /*
228     read operation
229     */
230     FILE* fout = fopen(obj_fn,"a");
231     //fprintf(fout,"\nread:\n");
232     fprintf(fout, "    li $v0, 4\n");
233     fprintf(fout,"    la $a0, _prompt\n");
234     fprintf(fout,"    syscall\n");
235     fprintf(fout, "    li $v0, 5\n");
236     fprintf(fout, "    syscall\n");
237     fclose(fout);
238 }
```

若函数名为 put，则调用写函数，生成将给定地址中的值输出的汇编码。

```
395 void writeid(int addr){
396     string reg="$a0";
397     lw(reg,addr);
398     writeop();
399 }
```

```
381 void writeop(){
382     /*
383     write operation in asm code
384     */
385     FILE* fout = fopen(obj_fn,"a");
386     //fprintf(fout,"\nwrite:\n");
387     fprintf(fout, "    li $v0, 1\n");
388     fprintf(fout,"    syscall\n");
389     fprintf(fout,"    li $v0, 4\n");
390     fprintf(fout, "    la $a0, _ret\n");
391     fprintf(fout, "    syscall\n");
392     fprintf(fout, "    move $v0, $0\n");
393     fclose(fout);
394 }
```

各种运算的实现，类似于 Syntax-DirectedTranslation 中对表达式求值的过程，只不过把求值替换为了生成目标代码。在语义动作 8 中添加为表达式生成目标代码的动作。因为该动作步骤较多，为了翻译方案的简洁起见，直接用 genCode 代表整个生成代码的动作过程。具体步骤分为三步，首先把操作数的值根据地址从内存中读到寄存器中，然后使用寄存器进行相应运算，最后把得到的运算结果存入到目标地址指向的内存中。

```
string reg="$t1";
lw(reg,gptree[pare->neibs[1]].saddr);
reg = "$t0";
lw(reg,pare->iaddr);
FILE * fout = fopen(obj_fn,"a");
```

上图中为把操作数的值根据地址从内存中读到寄存器中。

```
fprintf(fout,"    seq $t0, $t0, $t1\n");
```

上图中为使用寄存器进行相应运算（以==运算为例）。

```
fclose(fout);
sw(reg,gptree[pare->neibs[3]].iaddr);
```

上图中为将运算结果存入到目标地址指向的内存中。

分支语句的实现。首先是 if-else 语句。要实现这类分支控制语句，主要是根据条件跳转到对应的标签处，这就涉及到判断条件时，目标标签还没有生成的问题。一种做法是，生成跳转语句时，把目标标签的位置先空出来，等到后面生成了目标标签后再回填前面的空位。这里采取另一种做法，生成跳转语句时，预先生成后面跳转目标处的标签，并将该标签以继承属性的形式传递给后面添加标签的非终结符，后面添加标签时，非终结符采用前面接收到的继承属性 ilabel 作为标签号。通过这种预分配标签的方式，达到了生成跳转语句时好像已经分配好了后面的目标标签的效果。

```
void beqz(string& reg,int label){
    FILE* fout= fopen(obj_fn,"a");
    fprintf(fout,(string("    beqz ")+reg+string(", label")+
        to_string(label)+string("\n")).c_str());
    fclose(fout);
}
```

上图中为生成 beqz 语句汇编码的辅助函数。

```
void j(int label){
    FILE* fout= fopen(obj_fn,"a");
    fprintf(fout,(string("    j ")+string("label")+to_string(label)+
        string("\n")).c_str());
    fclose(fout);
}
```

上图中为生成 j 语句汇编码的辅助函数。

```
case 17:{
    gptree[pare->neibs[10]].ilabel=labeln++;
    string reg="$t0";
    lw(reg,gptree[pare->neibs[2]].saddr);
    beqz(reg,gptree[pare->neibs[10]].ilabel);
}
break;
case 18:{
    gptree[pare->neibs[12]].ilabel=labeln++;
    j(gptree[pare->neibs[12]].ilabel);
}break;
case 19:{
    attachLabel(gptree[pare->neibs[10]].ilabel);
}break;
case 20:{
    attachLabel(gptree[pare->neibs[12]].ilabel);
}break;
case 21:{
```

上图中为实现 if-else 语句汇编的语义动作实现代码。

while 语句的实现类似于 if-else 语句，只是要在 while 循环的判断条件前加标签，在 while 循环结束后加标签。在执行到循环体最尾端时，无条件跳回到判断条件前的标签，当不满足判断条件时，跳到循环结束后的标签。

```
case 21:{
    gptree[pare->neibs[10]].ilabel=labeln++;
    string reg="$t0";
    lw(reg,gptree[pare->neibs[3]].saddr);
    beqz(reg,gptree[pare->neibs[10]].ilabel);
}break;
case 22:{
    j(gptree[pare->neibs[2]].ilabel);
}break;
case 23:
    attachLabel(gptree[pare->neibs[10]].ilabel);
break;
case 24:{
    gptree[pare->neibs[2]].ilabel=labeln++;
    attachLabel(gptree[pare->neibs[2]].ilabel);
}break;
default:
```

上图中为实现 while 语句汇编的语义动作实现代码。

三、结果及分析和源程序调试过程

1）简易计算器

```
//输入数据 num1,num2,op，根据 op 确定操作进行运算，最后输出运算结果 ans
int num1,num2,op,ans;
get(num1,num2,op);
if(op==0)
{
    ans = num1 + num2;
};
if(op==1)
{
    ans = num1 - num2;
};
if(op==2)
{
    ans = num1 & num2;
};
if(op==3)
{
    ans = num1 | num2;
};
```

```
    put(ans);
```

生成的汇编码：

```
    .data
    _prompt: .asciiz "Please input an integer:"
    _ret: .asciiz "\n"
    .text
        li $v0, 4
        la $a0, _prompt
        syscall
        li $v0, 5
        syscall
        sw $v0, -16($sp)
        li $v0, 4
        la $a0, _prompt
        syscall
        li $v0, 5
        syscall
        sw $v0, -20($sp)
        li $v0, 4
        la $a0, _prompt
        syscall
        li $v0, 5
        syscall
        sw $v0, -24($sp)
        li $t0, 0
        sw $t0, 0($sp)
        lw $t1, 0($sp)
        lw $t0, -24($sp)
        seq $t0, $t0, $t1
        sw $t0, -32($sp)
        lw $t0, -32($sp)
        beqz $t0, label0
        lw $t1, -20($sp)
        lw $t0, -16($sp)
        add $t0, $t0, $t1
        sw $t0, -36($sp)
        lw $t0, -36($sp)
        sw $t0, -28($sp)
        j label1
    label0:
    label1:
        li $t0, 1
```

```
    sw $t0, -4($sp)
    lw $t1, -4($sp)
    lw $t0, -24($sp)
    seq $t0, $t0, $t1
    sw $t0, -40($sp)
    lw $t0, -40($sp)
    beqz $t0, label2
    lw $t1, -20($sp)
    lw $t0, -16($sp)
    sub $t0, $t0, $t1
    sw $t0, -44($sp)
    lw $t0, -44($sp)
    sw $t0, -28($sp)
    j label3
label2:
label3:
    li $t0, 2
    sw $t0, -8($sp)
    lw $t1, -8($sp)
    lw $t0, -24($sp)
    seq $t0, $t0, $t1
    sw $t0, -48($sp)
    lw $t0, -48($sp)
    beqz $t0, label4
    lw $t1, -20($sp)
    lw $t0, -16($sp)
    and $t0, $t0, $t1
    sw $t0, -52($sp)
    lw $t0, -52($sp)
    sw $t0, -28($sp)
    j label5
label4:
label5:
    li $t0, 3
    sw $t0, -12($sp)
    lw $t1, -12($sp)
    lw $t0, -24($sp)
    seq $t0, $t0, $t1
    sw $t0, -56($sp)
    lw $t0, -56($sp)
    beqz $t0, label6
    lw $t1, -20($sp)
    lw $t0, -16($sp)
    or $t0, $t0, $t1
```

```
        sw $t0, -60($sp)
        lw $t0, -60($sp)
        sw $t0, -28($sp)
        j label7
    label6:
    label7:
        lw $a0, -28($sp)
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, _ret
        syscall
        move $v0, $0
```

运行结果：

```
Please input an integer:7 5 0

Reset: reset completed.

Please input an integer:7
Please input an integer:5
Please input an integer:0
12

-- program is finished running (dropped off bottom) --


Reset: reset completed.

Please input an integer:6
Please input an integer:8
Please input an integer:1
-2

-- program is finished running (dropped off bottom) --


Reset: reset completed.

Please input an integer:5
Please input an integer:6
Please input an integer:2
4

-- program is finished running (dropped off bottom) --


Reset: reset completed.

Please input an integer:9
Please input an integer:12
Please input an integer:3
13
```

上述结果符合预期，汇编码生成正确。

2) 跑马灯

```
//循环输入 op，改变输出结果 out，输入 0 则结束程序
int num0,num1,out,op;
num1 = 3333;
num2 = 6666;
num3 = 9999;
op = 1;
while(op>0)
```

```
{
    if(op==1)
    {
        out = num1;
       };
    if(op==2)
    {
        out = num2;
    };
    if(op==2)
    {
        out = num3;
    };
    put(out);
    get(op);
};
```

原始样例没有声明 num2，num3，无法通过编译，故无法运行。此外，out=num3 的分支进入条件与 out=num2 的完全相同，因此当 op==2 时，这两个分支均会被执行。这里给出补充声明 num2，num3，并将 out=num3 的分支进入条件修改为 op==3 后的测试效果。

生成的汇编码：

```
.data
_prompt: .asciiz "Please input an integer:"
_ret: .asciiz "\n"
.text
    li $t0, 3333
    sw $t0, 0($sp)
    lw $t0, 0($sp)
    sw $t0, -36($sp)
    li $t0, 6666
    sw $t0, -4($sp)
    lw $t0, -4($sp)
    sw $t0, -32($sp)
    li $t0, 9999
    sw $t0, -8($sp)
    lw $t0, -8($sp)
    sw $t0, -40($sp)
    li $t0, 1
    sw $t0, -12($sp)
    lw $t0, -12($sp)
    sw $t0, -48($sp)
```

```
label0:
    li $t0, 0
    sw $t0, -16($sp)
    lw $t1, -16($sp)
    lw $t0, -48($sp)
    sgt $t0, $t0, $t1
    sw $t0, -52($sp)
    lw $t0, -52($sp)
    beqz $t0, label1
    li $t0, 1
    sw $t0, -20($sp)
    lw $t1, -20($sp)
    lw $t0, -48($sp)
    seq $t0, $t0, $t1
    sw $t0, -56($sp)
    lw $t0, -56($sp)
    beqz $t0, label2
    lw $t0, -36($sp)
    sw $t0, -44($sp)
    j label3
label2:
label3:
    li $t0, 2
    sw $t0, -24($sp)
    lw $t1, -24($sp)
    lw $t0, -48($sp)
    seq $t0, $t0, $t1
    sw $t0, -60($sp)
    lw $t0, -60($sp)
    beqz $t0, label4
    lw $t0, -32($sp)
    sw $t0, -44($sp)
    j label5
label4:
label5:
    li $t0, 3
    sw $t0, -28($sp)
    lw $t1, -28($sp)
    lw $t0, -48($sp)
    seq $t0, $t0, $t1
    sw $t0, -64($sp)
    lw $t0, -64($sp)
    beqz $t0, label6
    lw $t0, -40($sp)
```

```
        sw $t0, -44($sp)
        j label7
    label6:
    label7:
        lw $a0, -44($sp)
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, _ret
        syscall
        move $v0, $0
        li $v0, 4
        la $a0, _prompt
        syscall
        li $v0, 5
        syscall
        sw $v0, -48($sp)
        j label0
    label1:
```

运行结果：

```
3333
Please input an integer:1
3333
Please input an integer:2
6666
Please input an integer:3
9999
Please input an integer:4
9999
Please input an integer:2
6666
Please input an integer:6
6666
Please input an integer:5
6666
Please input an integer:1
3333
Please input an integer:3
9999
Please input an integer:0

-- program is finished running (dropped off bottom) --
```

上述结果符合预期，汇编码生成正确。

自定义测试样例：

单元测试：

1）get 函数参数非左值报错测试：

测试样例：

```
1  int a,b,c;
2  get(a,b,c);
3  get(a+b);
```

截图中仅展示了和 get 函数相关的部分。

测试结果：

```
Finished constructing parse tree!
inputs.c:3:5: error: expression must be an lvalue
inputs.c:6:6: error: conflicting declaration 'int a'
```

图中可以看到，成功对 3 行 5 列处 get 函数参数 a+b 非左值的问题进行了报错。

2）get 函数、put 函数功能测试：

测试样例：

```
1  int a,b,c;
2  get(a,b,c);
3  get(a+b);
4  put(c);
5  put(b);put(a);
```

截图中仅展示了和 get、put 函数相关的部分。

测试结果：

生成的汇编码：

```
.data
_prompt: .asciiz "Please input an integer:"
_ret: .asciiz "\n"
.text
    li $v0, 4
    la $a0, _prompt
    syscall
```

```
li $v0, 5
syscall
sw $v0, -36($sp)
li $v0, 4
la $a0, _prompt
syscall
li $v0, 5
syscall
sw $v0, -40($sp)
li $v0, 4
la $a0, _prompt
syscall
li $v0, 5
syscall
sw $v0, -44($sp)
lw $a0, -44($sp)
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
lw $a0, -40($sp)
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
```

lw $a0, -36($sp)

li $v0, 1

syscall

li $v0, 4

la $a0, _ret

syscall

move $v0, $0

Mars 上运行效果：

```
Please input an integer:1
Please input an integer:2
Please input an integer:3
3
2
1

-- program is finished running (dropped off bottom) --
```

效果符合预期，get、put 函数正确实现。

3）运算符功能测试

由于运算符太多，这里只选取几个较为典型的运算符进行测试。

测试用例：

```
int a,b,d;
_Bool c;
get(a,b);
get(a+b);
c=a==b;
put(c);
c=a||b;
put(c);
c=a&&b;
put(c);
c=a>b;
put(c);
d=a+b;
put(d);
d=a/b;
put(d);
```

生成的汇编码如下：

```
.data
_prompt: .asciiz "Please input an integer:"
_ret: .asciiz "\n"
.text
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    sw $v0, -36($sp)
    li $v0, 4
```

```
la $a0, _prompt
syscall
li $v0, 5
syscall
sw $v0, -40($sp)
lw $t1, -40($sp)
lw $t0, -36($sp)
add $t0, $t0, $t1
sw $t0, -52($sp)
lw $t1, -40($sp)
lw $t0, -36($sp)
seq $t0, $t0, $t1
sw $t0, -56($sp)
lw $t0, -56($sp)
sw $t0, -48($sp)
lw $a0, -48($sp)
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
lw $t1, -40($sp)
lw $t0, -36($sp)
or $t0, $t0, $t1
sgt $t0, $t0, 0
sw $t0, -60($sp)
lw $t0, -60($sp)
sw $t0, -48($sp)
```

```
lw $a0, -48($sp)
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
lw $t1, -40($sp)
lw $t0, -36($sp)
sgt $t0, $t0, $0
sgt $t1, $t1, $0
and $t0, $t0, $t1
sw $t0, -64($sp)
lw $t0, -64($sp)
sw $t0, -48($sp)
lw $a0, -48($sp)
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
lw $t1, -40($sp)
lw $t0, -36($sp)
sgt $t0, $t0, $t1
sw $t0, -68($sp)
lw $t0, -68($sp)
sw $t0, -48($sp)
lw $a0, -48($sp)
```

```
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
lw $t1, -40($sp)
lw $t0, -36($sp)
add $t0, $t0, $t1
sw $t0, -72($sp)
lw $t0, -72($sp)
sw $t0, -44($sp)
lw $a0, -44($sp)
li $v0, 1
syscall
li $v0, 4
la $a0, _ret
syscall
move $v0, $0
lw $t1, -40($sp)
lw $t0, -36($sp)
div $t0, $t0, $t1
sw $t0, -76($sp)
lw $t0, -76($sp)
sw $t0, -44($sp)
lw $a0, -44($sp)
li $v0, 1
syscall
li $v0, 4
```

la $a0, _ret

syscall

move $v0, $0

仿真运行结果如下：

```
Please input an integer:10
Please input an integer:5
0
1
1
1
15
2

-- program is finished running (dropped off bottom) --
```

上述运算实现正确。

4）if-else 分支测试

测试用例如下。

```
int a,b,d;
_Bool c;
get(a,b);
get(a+b);
c=a==b;
put(c);
if(c){
      d=a-b;
      put(d);
};else{
      d=a&b;
      put(d);
};
d=a+b;
put(d);
```

生成的汇编码如下。

```
.data
_prompt: .asciiz "Please input an integer:"
_ret: .asciiz "\n"
.text
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    sw $v0, 0($sp)
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    sw $v0, -4($sp)
    lw $t1, -4($sp)
    lw $t0, 0($sp)
    add $t0, $t0, $t1
    sw $t0, -16($sp)
    lw $t1, -4($sp)
    lw $t0, 0($sp)
    seq $t0, $t0, $t1
    sw $t0, -20($sp)
    lw $t0, -20($sp)
    sw $t0, -12($sp)
    lw $a0, -12($sp)
    li $v0, 1
```

```
        syscall
        li $v0, 4
        la $a0, _ret
        syscall
        move $v0, $0
        lw $t0, -12($sp)
        beqz $t0, label0
        lw $t1, -4($sp)
        lw $t0, 0($sp)
        sub $t0, $t0, $t1
        sw $t0, -24($sp)
        lw $t0, -24($sp)
        sw $t0, -8($sp)
        lw $a0, -8($sp)
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, _ret
        syscall
        move $v0, $0
        j label1
label0:
        lw $t1, -4($sp)
        lw $t0, 0($sp)
        and $t0, $t0, $t1
        sw $t0, -28($sp)
        lw $t0, -28($sp)
        sw $t0, -8($sp)
        lw $a0, -8($sp)
```

```
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
label1:
    lw $t1, -4($sp)
    lw $t0, 0($sp)
    add $t0, $t0, $t1
    sw $t0, -32($sp)
    lw $t0, -32($sp)
    sw $t0, -8($sp)
    lw $a0, -8($sp)
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
```

运行结果如下。

```
Please input an integer:10
Please input an integer:10
1
0
20

-- program is finished running (dropped off bottom) --
```

上图中为跳转条件为真时的结果。

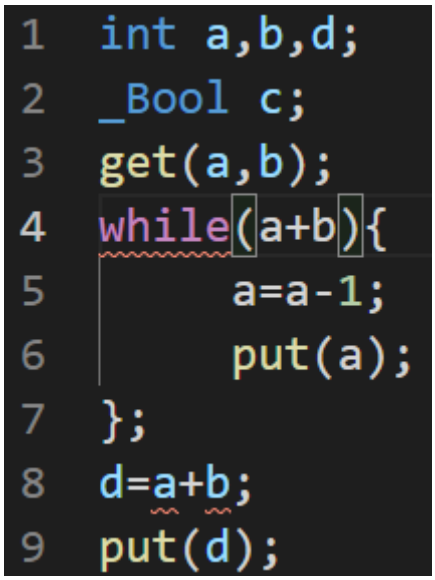
```
Please input an integer:21
Please input an integer:5
0
5
26

-- program is finished running (dropped off bottom) --
```

上图中为跳转条件为假时的结果。

综上可见，if-else 语句实现正确。

5）while 循环测试：

测试用例：



```
int a,b,d;
_Bool c;
get(a,b);
while(a+b){
        a=a-1;
        put(a);
};
d=a+b;
put(d);
```

生成的汇编码如下：

```
.data
_prompt: .asciiz "Please input an integer:"
_ret: .asciiz "\n"
.text
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    sw $v0, -4($sp)
```

```
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    sw $v0, -8($sp)
label0:
    lw $t1, -8($sp)
    lw $t0, -4($sp)
    add $t0, $t0, $t1
    sw $t0, -20($sp)
    lw $t0, -20($sp)
    beqz $t0, label1
    li $t0, 1
    sw $t0, 0($sp)
    lw $t1, 0($sp)
    lw $t0, -4($sp)
    sub $t0, $t0, $t1
    sw $t0, -24($sp)
    lw $t0, -24($sp)
    sw $t0, -4($sp)
    lw $a0, -4($sp)
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
    j label0
```

```
label1:
    lw $t1, -8($sp)
    lw $t0, -4($sp)
    add $t0, $t0, $t1
    sw $t0, -28($sp)
    lw $t0, -28($sp)
    sw $t0, -12($sp)
    lw $a0, -12($sp)
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
```

Mars 上运行结果如下：

```
Please input an integer:3
Please input an integer:5
2
1
0
-1
-2
-3
-4
-5
0

-- program is finished running (dropped off bottom) --
```

自定义测试样例：

```
int a,b,c=0,d;
get(a,b);
get(a+b);
while(a+b){
        a=a-1;
        put(a);
        if(c){
                d=a-b;
                put(d);
        };else{
                d=a&b;
                put(d);
        };
        c=1-c;
};
d=a*b;
put(d);
```

生成的汇编码：

```
.data
_prompt: .asciiz "Please input an integer:"
_ret: .asciiz "\n"
.text
    li $t0, 0
    sw $t0, 0($sp)
    lw $t0, 0($sp)
    sw $t0, -20($sp)
    li $v0, 4
    la $a0, _prompt
```

```
    syscall
    li $v0, 5
    syscall
    sw $v0, -12($sp)
    li $v0, 4
    la $a0, _prompt
    syscall
    li $v0, 5
    syscall
    sw $v0, -16($sp)
    lw $t1, -16($sp)
    lw $t0, -12($sp)
    add $t0, $t0, $t1
    sw $t0, -28($sp)
label0:
    lw $t1, -16($sp)
    lw $t0, -12($sp)
    add $t0, $t0, $t1
    sw $t0, -32($sp)
    lw $t0, -32($sp)
    beqz $t0, label1
    li $t0, 1
    sw $t0, -4($sp)
    lw $t1, -4($sp)
    lw $t0, -12($sp)
    sub $t0, $t0, $t1
    sw $t0, -36($sp)
    lw $t0, -36($sp)
    sw $t0, -12($sp)
```

```
    lw $a0, -12($sp)
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
    lw $t0, -20($sp)
    beqz $t0, label2
    lw $t1, -16($sp)
    lw $t0, -12($sp)
    sub $t0, $t0, $t1
    sw $t0, -40($sp)
    lw $t0, -40($sp)
    sw $t0, -24($sp)
    lw $a0, -24($sp)
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
    j label3
label2:
    lw $t1, -16($sp)
    lw $t0, -12($sp)
    and $t0, $t0, $t1
    sw $t0, -44($sp)
    lw $t0, -44($sp)
```

```
        sw $t0, -24($sp)
        lw $a0, -24($sp)
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, _ret
        syscall
        move $v0, $0
label3:
        li $t0, 1
        sw $t0, -8($sp)
        lw $t1, -20($sp)
        lw $t0, -8($sp)
        sub $t0, $t0, $t1
        sw $t0, -48($sp)
        lw $t0, -48($sp)
        sw $t0, -20($sp)
        j label0
label1:
        lw $t1, -16($sp)
        lw $t0, -12($sp)
        mul $t0, $t0, $t1
        sw $t0, -52($sp)
        lw $t0, -52($sp)
        sw $t0, -24($sp)
        lw $a0, -24($sp)
        li $v0, 1
        syscall
        li $v0, 4
```

```
la $a0, _ret
syscall
move $v0, $0
```

运行结果：

```
Please input an integer:3
Please input an integer:5
2
0
1
-4
0
0
-1
-6
-2
4
-3
-8
-4
4
-5
-10
-25

-- program is finished running (dropped off bottom) --
```

上述样例通过 c=1-c，且初始化 c=0，实现每次循环进入不同的 if 分支。以上结果完全符合预期，功能实现正确。

四、总结

1、遇到的问题及解决办法

问题 1：测试 get 函数功能时，不对非左值参数报错。

测试样例如下（只展示了和 get 函数相关的前三行）。

```
1  int a,b,c;
2  get(a,b,c);
3  get(a+b);
```

测试结果如下。

```
E:\programs\Language\VSCode\C++\syntax_analyzer.exe
E , expr 15 E | \e
15 \e
ST struct id 2 { ST' } ;
2 \e
ST' T ST' | ST ST' | ; ST' | \e
end
Begin to construct parse tree!
SYNTAX ERROR at Ln 4, Col 5! Unexpected symbol "("!
SYNTAX ERROR at Ln 4, Col 7! Unexpected symbol ")"!
Semantic Error at Ln 9, Col 7! Operator lost!
Semantic Error at Ln 10, Col 8! Operand lost!
Semantic Error at Ln 11, Col 9! ')' lost!
Finished constructing parse tree!
inputs.c:4:6: error: conflicting declaration 'int a'
inputs.c:4:9: error: conflicting declaration 'int b'
inputs.c:4:10: error: conversion from 'int' to 'unknown' requested
k = 0
f = 1
g = 3
h = 10
a = 2
inputs.c:10:8: note:  mismatched types 'int' and 'unknown'
inputs.c:10:8: note:  only int operands can participate in arithmetic operations
inputs.c:10:3: error: conversion from 'unknown' to 'int' requested
inputs.c:11:3: error: conversion from 'unknown' to 'int' requested
inputs.c:12:11: error: 'd' was not declared in this scope
inputs.c:12:11: note:  mismatched types 'int' and 'unknown'
inputs.c:12:1: error: 'e' was not declared in this scope
inputs.c:13:1: error: 'x' was not declared in this scope
Press any key to continue . . .
```

可以看到，结果中没有对非左值参数 a+b 报错。

解决方案：检查发现，翻译方案出错。

```
expr_list expr E
E , expr 15 E | \e
```

先前的翻译方案只对出现在逗号后面的 expr 执行语义动作，而对第一个 expr 不做任何动作。a+b 属于第一个 expr，因此被忽略。

$$expr\_list \rightarrow expr\ 16\ E$$
$$E \rightarrow ,\ expr\ 15\ E \mid \varepsilon$$
$$15 \rightarrow \varepsilon$$
$$16 \rightarrow \varepsilon$$

修改翻译方案，使之对第一个 expr 也执行语义动作。

问题 2：问题 1 按照上述方案修改后仍未解决。

解决方案：检查发现，expr 推出 a+b 后，expr 的 sname 并未如预期保持缺省值“-1”，而是为直接产生加法的非终结符 I。于是发现预先设计的方案存在缺陷，I 上面的非终结符推出空，因此会把 I 的名字传上去。而且在语法分析时，为每个非终结符的名字都赋了值，因此永远不会出现非终结符的 sname 为缺省值的情况。可通过判断 expr 的 sname 是否在符号表中，来达到判断其是否产生左值的效果。

问题 3：问题 2 按上述方案修改后仍未解决。

```
inputs.c:2:7: error: expression must be an lvalue
inputs.c:2:9: error: expression must be an lvalue
inputs.c:4:6: error: conflicting declaration 'int a'
inputs.c:4:9: error: conflicting declaration 'int b'
```

出现报错，但不符合预期。

解决方案：发现前面没改全，只改了 expr 出现在逗号后面的情况。此外，判断符号表中是否有 expr 的 sname 这个条件忘记取反。

```
Finished constructing parse tree!
a
b
c
I
inputs.c:3:5: error: expression must be an lvalue
inputs.c:4:6: error: conflicting declaration 'int a'
inputs.c:4:9: error: conflicting declaration 'int b'
```

修改后问题解决。

问题 3：测试“==”运算符时，结果不对。

测试用例如下。

```
1  int a,b;
2  _Bool c;
3  get(a,b);
4  get(a+b);
5  c=a==b;
6  put(c);
7  put(b);put(a);
```

测试结果如下。

```
Reset: reset completed.

Please input an integer:10
Please input an integer:10
0
10
10

-- program is finished running (dropped off bottom) --
```

输出的第一个数预期为 1，实际为 0。

解决方案：查看生成的汇编码，发现没有生成判断 a==b 的对应汇编语句 seq。检查发现，生成 seq 的代码部分，写完后忘记关闭文件指针，导致写的 seq 语句没能保存。修改后出现 seq 语句。

```
 9      syscall
10      sw $v0, -36($sp)
11      li $v0, 4
12      la $a0, _prompt
13      syscall
14      li $v0, 5
15      syscall
16      sw $v0, -40($sp)
17      lw $t1, -40($sp)
18      lw $t0, -36($sp)
19      sw $t0, -48($sp)
20      lw $t1, -40($sp)
21      lw $t0, -36($sp)
22      seq $t0, $t0, $t1
23      sw $t0, -52($sp)
24      lw $a0, -44($sp)
25      li $v0, 1
26      syscall
27      li $v0, 4
28      la $a0, _ret
29      syscall
30      move $v0, $0
```

问题 4：问题 3 修复后，结果仍然不对。

解决方案：分析上图，发现 seq 的运算结果存入的是表达式 a==b 的地址-52，而 c 的地址是-44，即赋值运算没有做。添加语义动作为赋值运算生成对应汇编码后，问题解决。

```
Please input an integer:10
Please input an integer:10
1
10
10

-- program is finished running (dropped off bottom) --
```

问题 5：加法和除法测试结果不对。

测试用例如下。

```
1  int a,b;
2  _Bool c;
3  get(a,b);
4  get(a+b);
5  c=a==b;
6  put(c);
7  c=a||b;
8  put(c);
9  c=a&&b;
10 put(c);
11 c=a+b;
12 put(c);
13 c=a/b;
14 put(c);
```

测试结果如下。

```
Please input an integer:10
Please input an integer:5
0
1
1
1
1

-- program is finished running (dropped off bottom) --
```

解决方案：检查发现是测试用例设计有误，c 为_Bool 型变量，加、乘法运算存在类型转换错误。修改测试用例如下。

```
1  int a,b,d;
2  _Bool c;
3  get(a,b);
4  get(a+b);
5  c=a==b;
6  put(c);
7  c=a||b;
8  put(c);
9  c=a&&b;
10 put(c);
11 c=a>b;
12 put(c);
13 d=a+b;
14 put(d);
15 d=a/b;
16 put(d);
```

修改后问题解决。

```
Please input an integer:10
Please input an integer:5
0
1
1
1
15
2

-- program is finished running (dropped off bottom) --
```

问题 6：语句 a=a-1 执行错误，执行后 a 的值没有-1。

解决方案:分析发现，这是因为没有将常数的值存到内存中。修改语义动作 7，使其将常数的值保存到其对应内存地址中。修改后问题解决。

```
Please input an integer:3
Please input an integer:5
2

-- program is finished running (dropped off bottom) --
```

问题 7：实现 while 循环时，出现循环不能退出的问题。

测试用例如下。

```
1  int a,b,d;
2  _Bool c;
3  get(a,b);
4  while(a+b){
5        a=a-1;
6        put(a);
7  };
8  d=a+b;
9  put(d);
```

测试结果如下。

```
Please input an integer:3
Please input an integer:5
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-11
-12
-13
-14
-15
-16
-17
-18
-19
-20
-21
-22
-23
-24
-25
-26
-27
-28
-29
```

测试陷入死循环，上图中仅展示了部分截图。

解决方案：检查汇编码，发现循环条件中的 a+b 没有被重新运算，即标签 label0 打的位置不对，太靠后了。

```
    lw $t0, -4($sp)
    add $t0, $t0, $t1
    sw $t0, -20($sp)
label0:
    lw $t0, -20($sp)
    beqz $t0, label1
    li $t0, 1
    sw $t0, 0($sp)
    lw $t1, 0($sp)
    lw $t0, -4($sp)
    sub $t0, $t0, $t1
    sw $t0, -24($sp)
    lw $t0, -24($sp)
    sw $t0, -4($sp)
    lw $a0, -4($sp)
    li $v0, 1
    syscall
    li $v0, 4
    la $a0, _ret
    syscall
    move $v0, $0
    j label0
label1:
    lw $t1, -8($sp)
    lw $t0, -4($sp)
```

将 label0 提前到计算判断条件之前。问题解决。

```
Please input an integer:3
Please input an integer:5
2
1
0
-1
-2
-3
-4
-5
0

-- program is finished running (dropped off bottom) --
```