

# YatCC-AI

## 编译原理课程实验

---

中山大学

郑馥丹、张献伟

YatCC 团队

2025/2/27

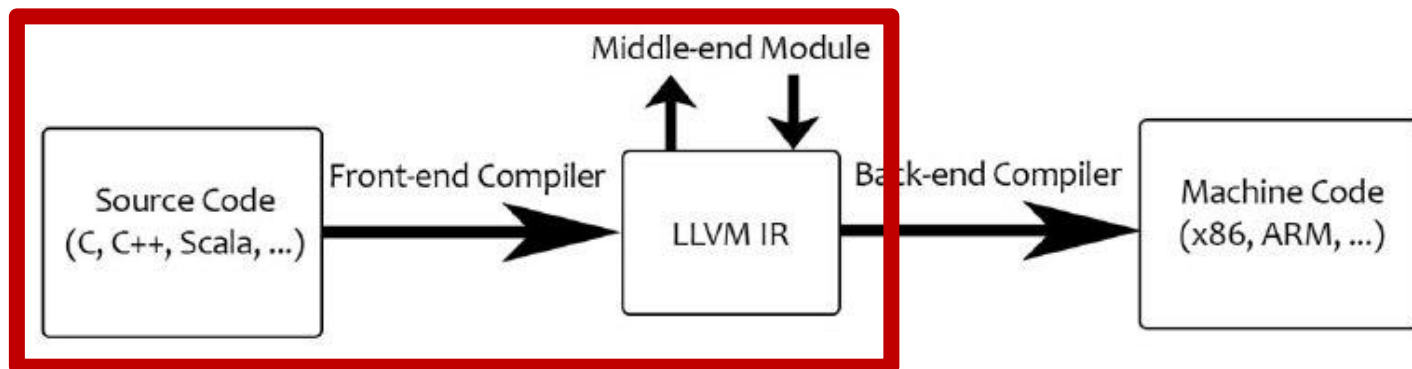


YAT COMPILER

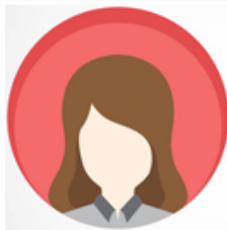
## ❖ YatCC是基于LLVM框架的编译器构造实验

在本课程中，你需要：

- 使用LLVM工业级编译器基础设施开发一个C语言子集的编译器
- 完成从C语言源代码到LLVM中间表示（甚至是汇编代码）的转换过程
- 预估编码总量：**3000LOC**左右



## 教师



郑馥丹

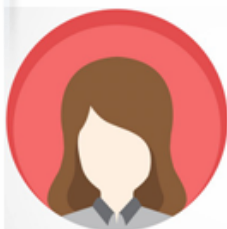


张献伟

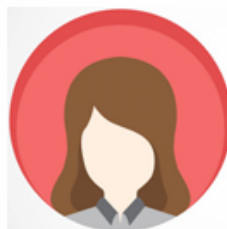


陈志广

## 助教



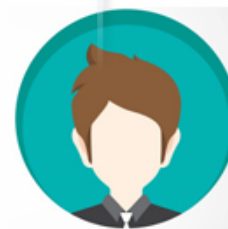
吴露



何静仪



顾宇浩



许宏鑫



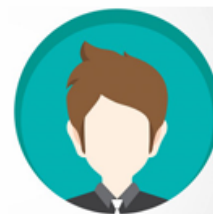
黄瀚



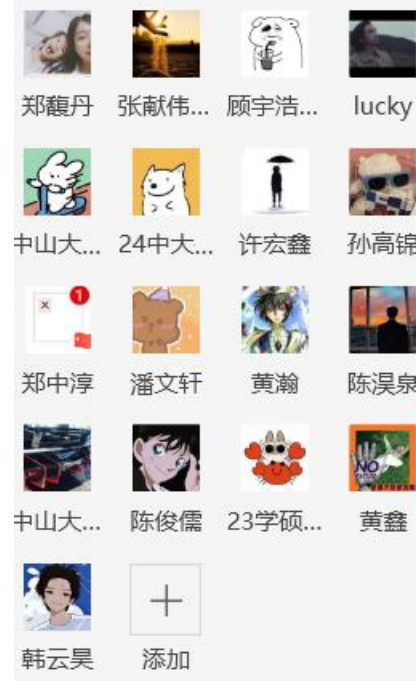
潘文轩



郑腾扬



孙高锦



❖ <https://yatcc-ai.com/ol.html>

- VS Code在线开发平台，基于我们自己的超算
- “配环境”的终极解决方案就是：不用配环境



HPC+AI多模式融合应用软件平台



## ❖ 点击即用的一站式在线平台

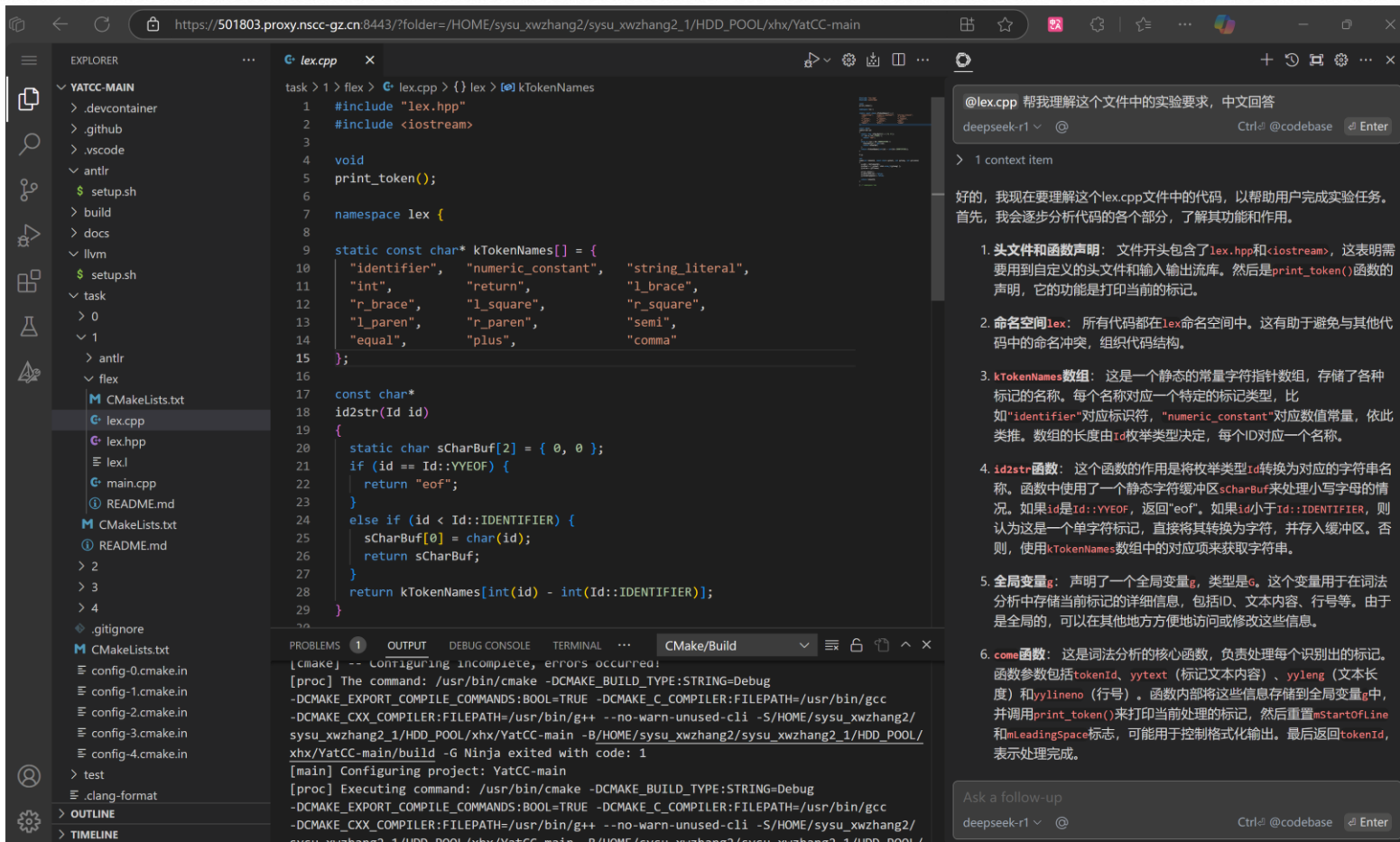
➤ <https://yatcc-ai.com/ol.html>

## ❖ 开箱即用的容器镜像

➤ `docker pull`  
`registry.cn-guangzhou.aliyuncs.com/`  
`yhgu2000/yatcc:latest`

## ❖ 配置自动化的代码仓库

➤ `git clone`  
`https://github.com/arcsysu/YatCC.git`



The screenshot displays a code editor with a C++ file named `lex.cpp`. The code defines a lexer for a simple language, including token names and a function to convert token IDs to strings. A chat window on the right shows a user asking for help understanding the code, and the AI providing a detailed explanation of the code's components.

**@lex.cpp 帮我理解这个文件中的实验要求，中文回答**

deepseek-r1 @ Ctrl @ codebase Enter

> 1 context item

好的，我现在要理解这个lex.cpp文件中的代码，以帮助用户完成实验任务。首先，我会逐步分析代码的各个部分，了解其功能和作用。

- 头文件和函数声明：** 文件开头包含了 `lex.hpp` 和 `<iostream>`，这表明需要用到自定义的头文件和输入输出流。然后是 `print_token()` 函数的声明，它的功能是打印当前的标记。
- 命名空间lex：** 所有代码都在 `lex` 命名空间中。这有助于避免与其他代码中的命名冲突，组织代码结构。
- kTokenNames数组：** 这是一个静态的常量字符串指针数组，存储了各种标记的名称。每个名称对应一个特定的标记类型，比如“identifier”对应标识符，“numeric\_constant”对应数值常量，依此类推。数组的长度由 `Id` 枚举类型决定，每个ID对应一个名称。
- id2str函数：** 这个函数的作用是将枚举类型 `Id` 转换为对应的字符串名称。函数中使用了一个静态字符缓冲区 `sCharBuf` 来处理小写字母的情况。如果 `id` 是 `Id::YYEOF`，返回“eof”；如果 `id` 小于 `Id::IDENTIFIER`，则认为这是一个单字符标记，直接将其转换为字符，并存入缓冲区。否则，使用 `kTokenNames` 数组中的对应项来获取字符串。
- 全局变量g：** 声明了一个全局变量 `g`，类型是 `G`。这个变量用于在词法分析中存储当前标记的详细信息，包括ID、文本内容、行号等。由于是全局的，可以在其他地方方便地访问或修改这些信息。
- come函数：** 这是词法分析的核心函数，负责处理每个识别出的标记。函数参数包括 `tokenId`、`yytext`（标记文本内容）、`yylen`（文本长度）和 `yylineno`（行号）。函数内部将这些信息存储到全局变量 `g` 中，并调用 `print_token()` 来打印当前处理的标记，然后重置 `mStartOfLine` 和 `mLeadingSpace` 标志，可能用于控制格式化输出。最后返回 `tokenId`，表示处理完成。

Ask a follow-up  
deepseek-r1 @ Ctrl @ codebase Enter

\*由于大模型需求火爆，中心资源紧张，  
AI服务会在课程过程中逐渐普及上线……



deepseek

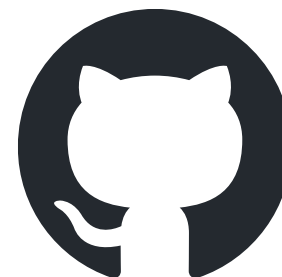
## ❖ Bilibili 教程

➤ <https://space.bilibili.com/3546650047941291>



## ❖ Github 文档

➤ <https://arcsysu.github.io/YatCC>



## ❖ LLVM 官方文档

➤ <https://llvm.org/docs/>





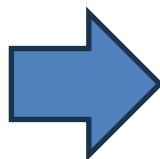




# 任务与考核

---

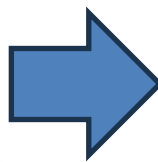
```
Yuhao Gu, 12 months ago | 1 author (Yuhao Gu)
1  int main(){
2      return 3;
3  }
```



```
# 1 "./basic/000_main.sysu.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 384 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "./basic/000_main.sysu.c" 2
int main(){
    return 3;
}
```

- 使用flex/antlr读取源代码，解析成TOKEN流
- 评分标准：正确识别TOKEN的**类型与位置**并顺序输出

```
# 1 "./basic/000_main.sysu.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 384 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "./basic/000_main.sysu.c" 2
int main(){
    return 3;
}
```



int 'int'

[StartOfLine] Loc=<./basic/000\_main.s

identifier 'main' [LeadingSpace] Loc=<./basic/0

l\_paren '(' Loc=<./basic/000\_main.sysu.c:1:9>

r\_paren ')' Loc=<./basic/000\_main.sysu.c:1:10>

l\_brace '{' Loc=<./basic/000\_main.sysu.c:1:11>

return 'return' [StartOfLine] [LeadingSpace] Lo

numeric\_constant '3' [LeadingSpace] Loc=<./basi

semi ';' Loc=<./basic/000\_main.sysu.c:2:13>

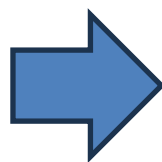
r\_brace '}' [StartOfLine] Loc=<./basic/000\_main

eof '' Loc=<./basic/000\_main.sysu.c:3:2>

```

int 'int'      [StartOfLine]    Loc=<./basic/000_main.s
identifier 'main'    [LeadingSpace]    Lo
l_paren '('      Loc=<./basic/000_main.s
r_paren ')'      Loc=<./basic/000_main.s
l_brace '{'      Loc=<./basic/000_main.s
return 'return'    [StartOfLine] [Leading
numeric_constant '3'    [LeadingSpace]
semi ';'          Loc=<./basic/000_main.syste
r_brace '}'      [StartOfLine]    Loc=<./ba
eof ''           Loc=<./basic/000_main.syste
  
```

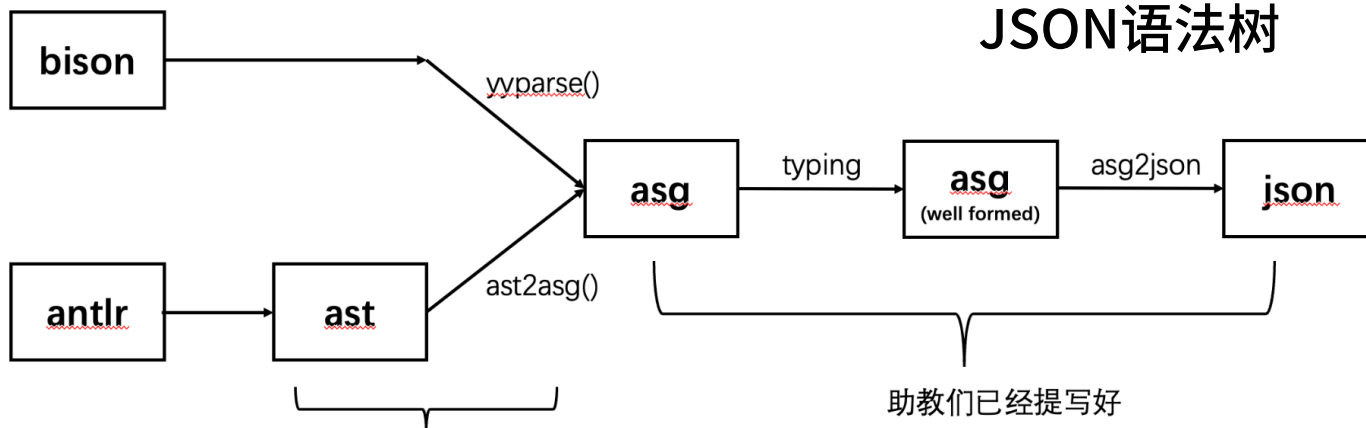
token流



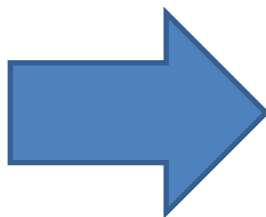
```

19      "name": "__int128_t",
20 >    "type": { ...
22      },
23      "inner": [
24        {
25          "id": "0x55c47643ae90",
26          "kind": "BuiltinType",
27 >    "type": { ...
29        }
30      ]
31    },
32    {
33      "id": "0x55c47643b168",
34      "kind": "TypedefDecl",
35      "loc": {},
36      "range": { ...
37 >    },
40    },
41    "isImplicit": true,
42    "name": "__uint128_t",
  
```

JSON语法树



需要同学们填空



```

3  "kind": "TranslationUnitDecl",
4  "loc": {},
5  > "range": { ...
8  },
9  "inner": [
10   {
11     "id": "0x55c47643b0f8",
12     "kind": "TypedefDecl",
13     "loc": {},
14   > "range": { ...
17   },
18   "isImplicit": true,
19   "name": "__int128_t",
20   > "type": { ...
22   },
23   "inner": [
24     {
25       "id": "0x55c47643ae90",
26       "kind": "BuiltinType",
27     > "type": { ...
29     }
30   }
31   ]
32 },
33 {
34   "id": "0x55c47643b168",
35   "kind": "TypedefDecl",
36   "loc": {},
37   > "range": { ...
40   },
41   "isImplicit": true,
42   "name": "__uint128_t",

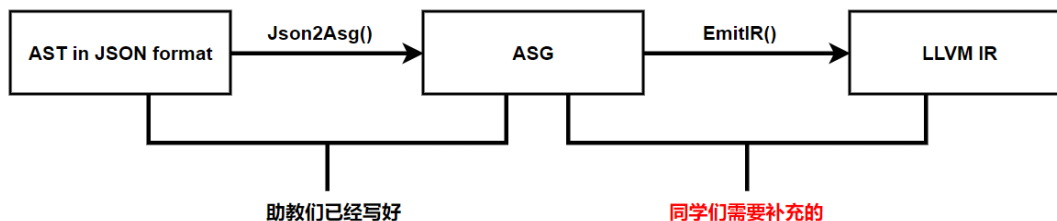
```

```

6  ; Function Attrs: noinline nounwind optnone
   ▷ Run | ⚙ Debug
7  define dso_local i32 @main() #0 {
8  entry:
9      %retval = alloca i32, align 4
10     store i32 0, ptr %retval, align 4
11     ret i32 3
12 }

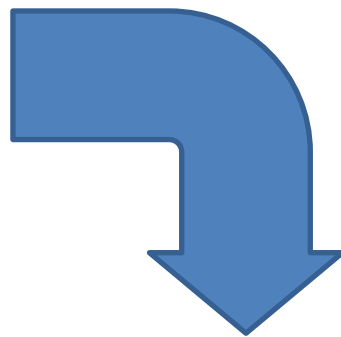
```

- 使用LLVM库，将JSON语法树转换为LLVM IR
- 解析ASG的代码已由助教完成，节省工作量！
- 评分标准：生成的LLVM IR能正确编译运行





## 性能优化



- 遵照LLVM框架要求，编写LLVM Pass，优化LLVM IR
- 评分标准：你的程序和clang-O2的相对性能

```

27 ; Function Attrs: noline nounwind optnone
    > Run | > Debug
28 define dso_local i32 @main() #0 {
29     entry:
30         %retval = alloca i32, align 4
31         %sum = alloca i32, align 4
32         %i = alloca i32, align 4
33         %tmp = alloca i32, align 4
34         %j = alloca i32, align 4
35         store i32 0, ptr %retval, align 4
36         store i32 0, ptr %sum, align 4
37         store i32 0, ptr %i, align 4
38         %call = call i32 (...) @_sys_getint()
39         store i32 %call, ptr @loopCount, align 4
40         call void @_sys_starttime(i32 noundef 21)
41         br label %while.cond
42
43 while.cond:
44     %0 = load i32, ptr %i, align 4
45     %1 = load i32, ptr @loopCount, align 4
46     %cmp = icmp slt i32 %0, %1
47     br i1 %cmp, label %while.body, label %while.end7
48
49 while.body:
50     store i32 0, ptr %tmp, align 4
51     store i32 0, ptr %j, align 4
52     br label %while.cond1
53
54 while.cond1:
55     %2 = load i32, ptr %j, align 4
56     %cmp2 = icmp slt i32 %2, 6
57     br i1 %cmp2, label %while.body3, label %while.end
58
59 while.body3:
60     %3 = load i32, ptr %i, align 4
61     call void @func(i32 noundef %3)
62     %4 = load i32, ptr %tmp, align 4
63     %5 = load i32, ptr @global, align 4
64     %add = add nsw i32 %4, %5

```

```

16 ; Function Attrs: nounwind
    > Run | > Debug
17 define dso_local noundef i32 @main() local_unnamed_addr #1 {
18     entry:
19         %call = tail call i32 (...) @_sys_getint() #3
20         store i32 %call, ptr @loopCount, align 4, !tbaa !2
21         tail call void @_sys_starttime(i32 noundef 21) #3
22         %0 = load i32, ptr @loopCount, align 4, !tbaa !2
23         %cmp17 = icmp sgt i32 %0, 0
24         br i1 %cmp17, label %while.cond1.preheader, label %while.end7
25
26 while.cond1.preheader:
27     %indvars.iv = phi i32 [ %indvars.iv.next, %while.cond1.preheader ], [ 0, %entry ]
28     %i.019 = phi i32 [ %add6, %while.cond1.preheader ], [ 0, %entry ]
29     %sum.018 = phi i32 [ %rem, %while.cond1.preheader ], [ 0, %entry ]
30     %div = udiv i32 %indvars.iv, 6
31     %add5 = add nsw i32 %div, %sum.018
32     %rem = srem i32 %add5, 134209537
33     %add6 = add nuw nsw i32 %i.019, 1
34     %indvars.iv.next = add i32 %indvars.iv, 6
35     %exitcond.not = icmp eq i32 %add6, %0
36     br i1 %exitcond.not, label %while.cond.while.end7_crit_edge, label %while.cond1.preheader, !llvm.loop !6
37
38 while.cond.while.end7_crit_edge:
39     %1 = add nsw i32 %0, -1
40     store i32 %1, ptr @global, align 4, !tbaa !2
41     br label %while.end7
42
43 while.end7:
44     %sum.0.lcssa = phi i32 [ %rem, %while.cond.while.end7_crit_edge ], [ 0, %entry ]
45     tail call void @_sys_stoptime(i32 noundef 37) #3
46     tail call void @_sys_putint(i32 noundef %sum.0.lcssa) #3
47     tail call void @_sys_putch(i32 noundef 10) #3
48     ret i32 0
49 }

```

```

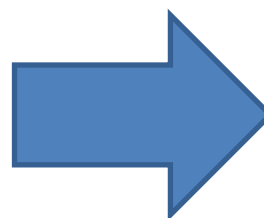
16 ; Function Attrs: nounwind
17 > Run | > Debug
18 define dso_local noundef i32 @main() local_unnamed_addr #1 {
19 entry:
20   %call = tail call i32 (...) @_sys_getint() #3
21   store i32 %call, ptr @loopCount, align 4, !tbaa !2
22   tail call void @_sys_starttime(i32 noundef 21) #3
23   %0 = load i32, ptr @loopCount, align 4, !tbaa !2
24   %cmp17 = icmp sgt i32 %0, 0
25   br i1 %cmp17, label %while.cond1.preheader, label %while.end7

while.cond1.preheader:
26   ; preds = %entry, %while.cond1.preheader
27   %indvars.iv = phi i32 [ %indvars.iv.next, %while.cond1.preheader ], [ 0, %entry ]
28   %i.019 = phi i32 [ %add6, %while.cond1.preheader ], [ 0, %entry ]
29   %sum.018 = phi i32 [ %rem, %while.cond1.preheader ], [ 0, %entry ]
30   %div = udiv i32 %indvars.iv, 6
31   %add5 = add nsw i32 %div, %sum.018
32   %rem = srem i32 %add5, 134209537
33   %add6 = add nuw nsw i32 %i.019, 1
34   %indvars.iv.next = add i32 %indvars.iv, 6
35   %exitcond.not = icmp eq i32 %add6, %0
36   br i1 %exitcond.not, label %while.cond.while.end7_crit_edge, label %while.cond1.preheader, !llvm.loop !6

while.cond.while.end7_crit_edge:
37   ; preds = %while.cond1.preheader
38   %1 = add nsw i32 %0, -1
39   store i32 %1, ptr @global, align 4, !tbaa !2
40   br label %while.end7

while.end7:
41   ; preds = %while.cond.while.end7_crit_edge, %entry
42   %sum.0.lcssa = phi i32 [ %rem, %while.cond.while.end7_crit_edge ], [ 0, %entry ]
43   tail call void @_sys_stoptime(i32 noundef 37) #3
44   tail call void @_sys_putint(i32 noundef %sum.0.lcssa) #3
45   tail call void @_sys_putch(i32 noundef 10) #3
46   ret i32 0
47 }

```



```

31 main:
32   pushq   %rbp
33   .seh_pushreg   %rbp
34   movq    %rsp, %rbp
35   .seh_setframe  %rbp, 0
36   subq    $48, %rsp
37   .seh_stackalloc 48
38   .seh_endprologue
39   call    __main
40   movl    $0, -4(%rbp)
41   movl    $0, -8(%rbp)
42   call    _sys_getint
43   movl    %eax, loopCount(%rip)
44   movl    $15, %ecx
45   call    _sys_starttime
46   jmp     .L4

.L7:
47   movl    $0, -12(%rbp)
48   movl    $0, -16(%rbp)
49   jmp     .L5

.L6:
50   movl    -8(%rbp), %eax
51   movl    %eax, %ecx
52   call    func
53   movl    global(%rip), %eax

```

- 使用LLVM库, 将LLVM IR转换为汇编指令
- 转换框架由助教提供
- 评分标准: 生成的汇编指令能正确编译运行



# 考勤 (10分)



# task0 演示&详解

---





# task0 (环境配置)



- 环境配置
- 熟悉开发工具
- 熟悉任务构建、评分流程
- 熟悉大模型工具
- 。 。 。



# 预祝大家实验顺利!

---

YatCC-AI  
编译原理课程实验  
2025/2/27

