问题及解决方法

ASG

1. 设计为struct，struct是public的，而class是private的。继承struct相对方便
2. 为了避免函数传参时大量的拷贝，例如vector，string，可以使用引用&，为了增加引用的参数能够接受的参数范围(左值，右值，常数)，需要增加常数引用const &
3. 增加默认构造函数，指针默认为nullptr

JsonToASG

1. 有关type
    1. tester/function_test2020/15_array_test3.sysu.c
       函数中的参数为 int a[][5]
       json qualtype为 `int (*)[5]`
       getType 单独增加处理 `(*)`
    2. tester/h_functional/100_int_literal.sysu.c
       const int k1 = 0x80000000;
       0x80000000 json qualtype: unsigned int
       变量应该考虑: is_const is_unsigned basictype ptr array `(*)`
       getType 在处理basictype之前须先处理is_unsigned
       不过is_unsigned实际上用不到，不会检查
    3. 函数参数要注意处理可变参数"..."
2. 库函数memmove冲突
   tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-public/conv0.sysu.c

```
if(object->getString("storageClass").getValueOr("").str() == "extern")
return nullptr;
```

3. 字符串
    - json会将源文件的字符原封不动的存入
      `const char r[9] = "\\\\\\\""`
      `json "value": "\"\\\\\\\\\\\\\\\\\"\""`
    - 需要处理的就是将被转义的字符`\`，与后续字符`\, n, ', "`，转化成在源文件中的转义字符`\\, \n, \', \"`
    - 注意c对单独的字符`\, ', "`的表示也是需要转义的

AGSToIR

1. 全局变量和局部变量的区别
    - 定义: 全局变量需要插入module的GlobalVariable，而局部变量则是插入符号表
    - 初始化: 如果是全局变量初始化，使用到了其他的全局变量，直接返回的是引用变量的初始值，即不会创建额外的load指令(隐式转换LValueToRValue)
2. alloca指令
   全部设置在entry block最前面，避免循环出错
3. and or

- 短路处理
  使用PHI指令，注意对于PHI节点值为rhs，跳转的块应该为当前正在插入的块，而不是rhs block。因为rhs block里面可能会产生其他的逻辑and or，使得插入块发生变化。
- 优先级
  对于||运算，一定先执行lhs，才能继续处理后续块，因为lhs可能为&&。

4. 隐式转换
   1. IntegralCast: int -> unsigned int
   2. FloatingCast: float -> double
   3. BitCast: `[270 x float]* -> float*`
      - tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-private/derich1.sysu.c

        ```
        /root/sysu/bin/sysu-optimizer: <stdin>:701:31: error: '@_sysy_getfarray' defined with type 'i32 (float*)*' but expected 'i32 ([270 x float]*)*'
        %_sysy_getfarray = call i32 @_sysy_getfarray([270 x float]* getelementptr inbounds ([512 x [270 x float]], [512 x [270 x float]]* @imgIn, i64 0, i64 0))
        ```

      - 对应源代码是将[270x512]的数组做参数，参数类型为float a[]，即`float*`
      - 此处会发生两次隐式转换
        (相同情况测试)

隐式转换需要增加bitcast的情况

5. if\while\do的end_block的处理
   最后在push_back到当前函数的BasicBlockList中
   因为if\while\do的body可能会产生额外的块，如果一开始将end_block就插入当前函数的BasicBlockList中，则body中产生的block是在end_block之后的，逻辑上是不对的。
6. if_then\if_else\while_body\do_body无条件分支跳转指令
   在这些body里都有可能产生额外的跳转指令，如果最后一条指令是跳转或者返回指令，则无需再增加无条件跳转指令。
   所以需要增加额外的函数getOrCreateBr来判断，是否增加Br指令。

多文件编译：

1. 在`CMakeLists.txt`中的`add_executable`增加新的实现文件:

```
    add_executable(sysu-generator main.cc JsonToASG.cpp EmitIR.cpp)
```

2. 头文件被多次include
   在每个头文件开始加上：

```
    #ifndef XX_HPP
    #define XX_HPP
```

   结尾加上`#endif`

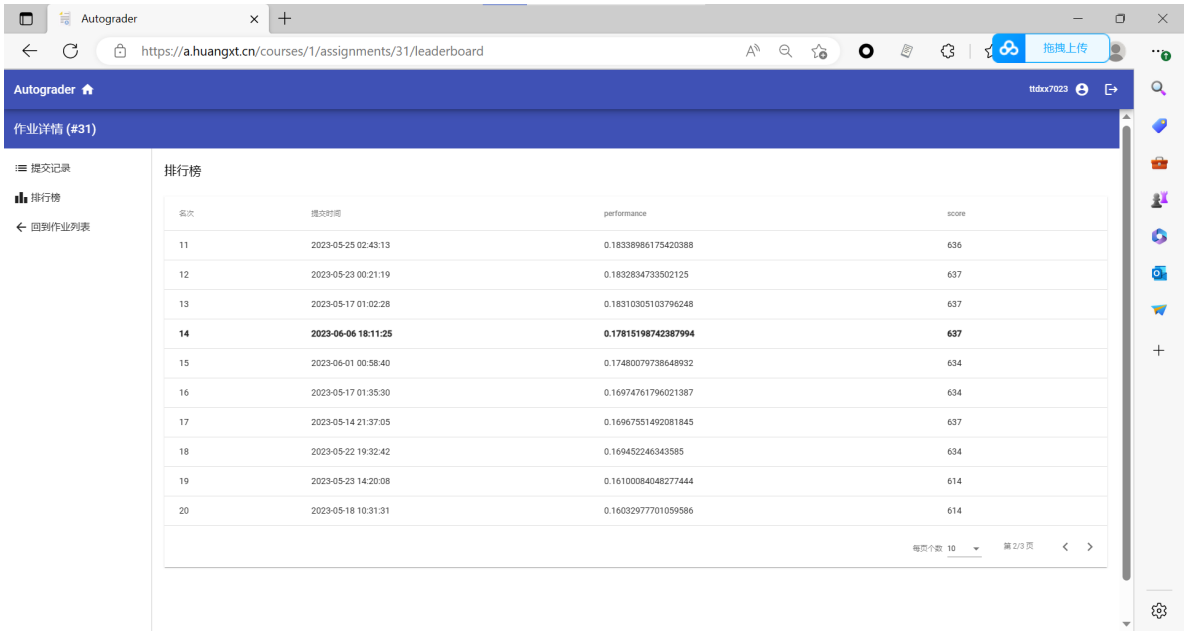
测评结果

1. 本地机timeout: 12个
    1. /workspace/SYsU-lang/tester/function_test2022/86_long_code2.sysu.c
    2. /workspace/SYsU-lang/tester/h_functional/107_long_code2.sysu.c
    3. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/dead-code-elimination-1.sysu.c
    4. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/dead-code-elimination-2.sysu.c
    5. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/dead-code-elimination-3.sysu.c
    6. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/hoist-3.sysu.c
    7. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/integer-divide-optimization-3.sysu.c
    8. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/dead-code-elimination-1.sysu.c
    9. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/dead-code-elimination-2.sysu.c
    10. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/dead-code-elimination-3.sysu.c
    11. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/hoist-3.sysu.c
    12. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/integer-divide-optimization-3.sysu.c
2. 测评机timeout: 10个
    1. /workspace/SYsU-lang/tester/function_test2022/86_long_code2.sysu.c

2. /workspace/SYsU-lang/tester/h_functional/107_long_code2.sysu.c
3. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/dead-code-elimination-3.sysu.c
4. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/hoist-3.sysu.c
5. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/integer-divide-optimization-2.sysu.c
6. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/integer-divide-optimization-3.sysu.c
7. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/dead-code-elimination-3.sysu.c
8. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/hoist-3.sysu.c
9. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/integer-divide-optimization-2.sysu.c
10. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/integer-divide-optimization-3.sysu.c

3. 两者有些不同，奇怪的是有本地机可以过的，但测评机却超时了
    1. tester/third_party/SYsU-lang-tester-perfermance/performance_test2021-private/integer-divide-optimization-2.sysu.c
    2. /workspace/SYsU-lang/tester/third_party/SYsU-lang-tester-perfermance/performance_test2022-public/integer-divide-optimization-2.sysu.c

排行榜截图



参考

1. LLVM系列第十五章：写一个简单的中间代码生成器IR Generator_飞翼剑仆的博客-CSDN博客
2. LLVM IR 快速上手 · GitBook (buaa-se-compiling.github.io)
3. Kaleidoscope: Code generation to LLVM IR — LLVM 11 documentation
4. A Complete Guide to LLVM for Programming Language Creators (mukulrathi.com)
5. SYsU-Lang: generate 快速上手 (yuque.com)