answer

> 在 Light IR 简介里，你已经了解了 IR 代码的基本结构，请尝试编写一个有全局变量的 cminus 程序，并用 clang 编译生成中间代码，解释全局变量在其中的位置。

cminus程序：

```
int global_test=0;
int main()
{
    return global_test+1;
}
```

运行得到.ll文件：

```
; ModuleID = 'global_test.c'
source_filename = "global_test.c"
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"

@global_test = dso_local global i32 0, align 4

; Function Attrs: noinline nounwind optnone uwtable
```

```
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %2 = load i32, i32* @global_test, align 4
  %3 = add nsw i32 %2, 1
  ret i32 %3
}

attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2, !3, !4}
!llvm.ident = !{!5}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{i32 7, !"uwtable", i32 1}
!4 = !{i32 7, !"frame-pointer", i32 2}
!5 = !{!"Ubuntu clang version 14.0.0-1ubuntu1.1"}

```

顶部的@global_test = dso_local global i32 0, align 4就是这个全局变量

> Light IR 中基本类型 label 在 Light IR C++ 库中是如何用类表示的？

BasicBlock类

如

```
auto lable_bb = BasicBlock::create(module, "lable",
mainFun);
```

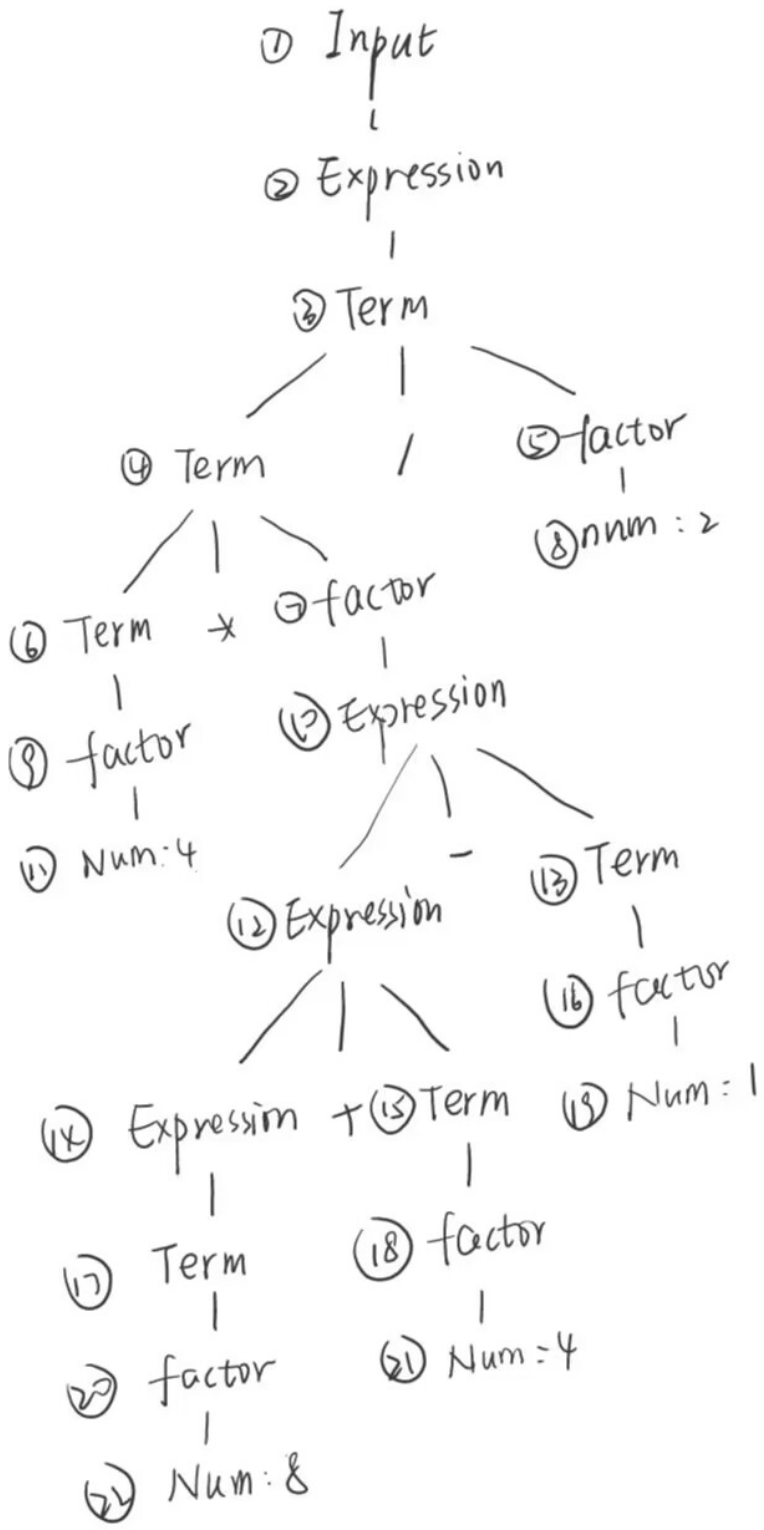
> Light IR C++ 库中 `Module` 类中对基本类型与组合类型存储的方式是一样的吗？请尝试解释组合类型使用其存储方式的原因。

不同

原因：组合类型不仅包括单一的数据值，还涉及到数据的组织关联等复杂性，需要更多的存储和描述信息。例如，结构体可能包含多个不同类型的成员，数组可能包含多个元素，指针可能引用其他数据

> 分析 `calc` 程序在输入为 `4 * (8 + 4 - 1) / 2` 时的行为：
>
> 1. 请画出该表达式对应的抽象语法树（使用 `calc_ast.hpp` 定义的语法树节点来表示，并给出节点成员存储的值），并给节点使用数字编号。

> **2.** 请给出示例代码在用访问者模式遍历该语法树时，访问者到达语法树节点的顺序。序列请按如下格式指明（序号为问题 1.a 中的编号）：3->2->5->1->1

- 

- 1->2->3->4->6->9->11->7->10->12->14->17->20->22->15->18->21->13->16->19->5->8