实验报告

代码结构设计

在 `src/executor` 中添加了如下 Class：

- `ActivationRecords`：活动记录的抽象类
  主要包含如下成员变量：

  ```
  var name: ?String
  var controlLink: ?ActivationRecords // 调用链
  var accessLink: ?ActivationRecords  // 静态链（闭包/静态作用域）
  let items: HashMap<String, Record> // 每一条记录
  ```

  这个抽象类中封装了对于沿 `accessLink` 添加、查找、更新变量和相应语义检查的方法，同时将 `node` 作为参数传递，方便报出错误位置
- `BlockActivationRecords`：块的活动记录的类，继承于 `ActivationRecords`
  目前还没有对 `ActivationRecords` 进行扩展
- `FuncActivationRecords`：函数的活动记录的类，继承于 `ActivationRecords`
  增加了如下成员变量：

  ```
  var return_addr : Node; // 返回节点
  var save_returnv_addr : Int64; // 保存返回值接收的地址，是活动记录中的 items 的索引
  var arguments : List<Record>; // 参数
  ```

  由于还没有实现函数调用，所以这里的设计还有待考量
- `Record`：每一条记录
  存放如下数据：

  ```
  var name: Token
  var value: ?Value
  var recordType: ?TokenKind // 类型
  var keyword: Token // var / let
  ```

表达式的实现：

- 作用域使用一个栈进行管理，栈中每一个元素是一个ActivationRecords，然后通过curRecords变量维护当前的作用域
- 二元运算符：
  写了许多形如这样的函数：
  `private func calcIntConstInt(a : Int64, b : Int64, op : TokenKind, node : Node) : Int64`
  表示接受两个 `int` 参数返回 `int`，然后在其中进行计算和语义检查
  在 visit BinaryExpr 的时候调用这些函数
- 一元运算符：简单实用模式匹配进行处理
- 条件表达式：通过计算 `cond` 的值选择访问的 `node`

- 循环表达式：[while、break、continue表达式的实现]
- 赋值：对活动记录进行修改，将未定义的错误识别封装在修改活动记录的方法中
- 变量定义：检查是否重复定义并修改活动记录

我认为的亮点：

- while、break、continue表达式：
  - `while` 在根据 `cond` 进行模拟时尝试 catch 块内的 `CjcjRuntimeErrorWithLocation` 并检查 `ErrorCode` 如果是 `BREAK_OUTSIDE_LOOP` 或 `CONTINUE_OUTSIDE_LOOP` 就进行对 `break` 和 `continue` 的模拟并修改程序的活动记录链，如果是其他的 `ErrorCode` 就继续抛出
  - `break` 和 `while` 的实现就直接抛出对应的 `CjcjRuntimeErrorWithLocation` 即可
  - 具体实现：

```
  public open override func visit(expr: WhileExpr): Value {
      var cond: Value = expr.condition.traverse(this);
      // 记录while外部的record
      var outterWhileRecordsNum: Int64 = recordsStack.size;

      match (cond) {
          case VBoolean(b) =>
              var the_cond: Bool = b;
              while (the_cond) {
                  try {
                      expr.block.traverse(this);
                  } catch (e: CjcjRuntimeErrorWithLocation) {
                      match (e.code) {
                          // 注意这里break和continue会导致出块，而出块需要更新recordsStack和
curRecords,就是将块直接更新到while外部的块
                          case ErrorCode.BREAK_OUTSIDE_LOOP =>
                              while (outterWhileRecordsNum < recordsStack.size) {
                                  recordsStack.remove()
                              }
                              curRecords = recordsStack.peek()
                              break; // break;
                          case ErrorCode.CONTINUE_OUTSIDE_LOOP =>
                              while (outterWhileRecordsNum < recordsStack.size) {
                                  recordsStack.remove()
                              }
                              curRecords = recordsStack.peek()
                              () // continue;
                          case _ => throw e;
                      }
                  }
    ...
```

遇到的问题和解决方案

> 遇到的问题：一开始使用一个全局变量和栈来维护 while 和 break continue 的对应关系，但是难以实现正确
> 解决：通过抛出异常的方式使得 `break` 和 `continue` 的信息能沿着"调用链"传播到第一个 `while`，同时还能跳过 `block` 中的剩余内容

> 遇到的问题：使用抛出异常的方式来实现 `break` 和 `continue` 会导致作用域不会正确地受到 `visitBlock` 中的维护（即 进入block作用域压栈->遍历所有语句->出block退栈，在这个过程中由于在遍历语句时会因异常跳出导致没有执行退栈操作）
> 解决：要对 `while` 做特殊处理，也就是在进入 `while` 前记录当前作用域，当捕获到 `break` 或 `continue` 的异常时要恢复作用域

> 遇到的问题：在处理字面量 `-9223372036854775808` 即 `Int64.min` 时，由于 unary 会将其拆解为 '-' 和 '9223372036854775808'，并在处理 '9223372036854775808' 时要将其存入 Int64变量 中会导致溢出
> 解决：只好对字面量 `-9223372036854775808` 在 unary 节点中做特殊处理

```
    public open override func visit(expr: UnaryExpr): Value {
        // 这里要对INT_MIN取负做出特殊处理
        match (expr.right) {
            case exp: LitConstExpr =>
                if (expr.oper.kind == TokenKind.SUB && exp.literal.value ==
"9223372036854775808") {
                    throw CjcjRuntimeErrorWithLocation(
                        ErrorCode.NEG_OVERFLOW,
                        "整数取负溢出",
                        expr
                    )
                }
            case _ => ()
        }
```

已知 bug

目前认为需要先遍历一遍ast构建accessLink，但是目前代码边解释边构建，在未来处理如函数闭包的时候可能会出bug，但是对于lab1来说不会。