Compiler2025 Lab1

王镜凯 22300240022

运行结果

- test1为例

```
------Parsing tmj source file: test1.tmj------------
Convert AST  to XML...
Saving AST (XML) to: test1.2.ast
Loading AST (XML) from: test1.2.ast
Saving AST (XML) to: test1.2-debug.ast
Loading AST (XML) from: test1.2-debug.ast
Clone it ...
Convert cloned AST to XML...
Saving cloned AST (XML) to: test1.2-debug3.ast
Rewriting AST...
Convert rewrote AST to XML...
Saving AST (XML) to: test1.2-debug4.ast
Applying constant propagation...
Convert constant propagated AST to XML...
Saving AST (XML) to: test1.2-debug5.ast
Executing the program...
Program returned: -7
-----Done---
```

- 除0测试

```
public int main() {
   x = ((-1)+((-2)*3));
   x = x;
   z = 3;
   z = (z+2);
   z = (3/0);
   return x;
}
```

```
------Parsing fmj source file: my_test3.fmj------------
Convert AST  to XML...
Saving AST (XML) to: my_test3.2.ast
Loading AST (XML) from: my_test3.2.ast
Saving AST (XML) to: my_test3.2-debug.ast
Loading AST (XML) from: my_test3.2-debug.ast
Clone it ...
Convert cloned AST to XML...
Saving cloned AST (XML) to: my_test3.2-debug3.ast
Rewriting AST...
Convert rewrote AST to XML...
Saving AST (XML) to: my_test3.2-debug4.ast
Applying constant propagation...
Error: Division by zero
terminate called after throwing an instance of 'std::runtime_error'
  what():  Executor Error: Division by zero
Aborted
```

> 在除0的地方进行了报错

- 没有return的测试

```
public int main() {
    x = ((-1)+((-2)*3));
    x = x;
    z = 3;
}
```

```
------Parsing fmj source file: my_test3.fmj------------
Convert AST  to XML...
Saving AST (XML) to: my_test3.2.ast
Loading AST (XML) from: my_test3.2.ast
Saving AST (XML) to: my_test3.2-debug.ast
Loading AST (XML) from: my_test3.2-debug.ast
Clone it ...
Convert cloned AST to XML...
Saving cloned AST (XML) to: my_test3.2-debug3.ast
Rewriting AST...
Convert rewrote AST to XML...
Saving AST (XML) to: my_test3.2-debug4.ast
Applying constant propagation...
Error: Division by zero
terminate called after throwing an instance of 'std::runtime_error'
  what():  Executor Error: Division by zero
Aborted
```

实现

> 参考MinusConverter、虎书、Copilot

ConstantPropagation

与MinusConverter相比较，主要在于对BinaryOp处理的不同，

```
void ConstantPropagation::visit(BinaryOp *node) {
  if (node == nullptr) {
    newNode = nullptr;
    return;
  }
  Exp *l = nullptr;
  if (node->left != nullptr) {
    node->left->accept(*this);
    l = static_cast<Exp *>(newNode);
  } else {
    cerr << "Error: No left expression found in the BinaryOp
statement" << endl;
    newNode = nullptr;
    return;
  }
  Exp *r = nullptr;
  if (node->right != nullptr) {
    node->right->accept(*this);
    r = static_cast<Exp *>(newNode);
  } else {
    cerr << "Error: No right expression found in the BinaryOp
statement"
         << endl;
    newNode = nullptr;
    return;
  }
  // Constant propagation logic
  if (l->getASTKind() == ASTKind::IntExp && r->getASTKind() ==
ASTKind::IntExp) {
    int leftVal = static_cast<IntExp *>(l)->val;
```

```
      int rightVal = static_cast<IntExp *>(r)->val;
      int result;
      if (node->op->op == "+") {
        result = leftVal + rightVal;
      } else if (node->op->op == "-") {
        result = leftVal - rightVal;
      } else if (node->op->op == "*") {
        result = leftVal * rightVal;
      } else if (node->op->op == "/") {
        if (rightVal == 0) {
          cerr << "Error: Division by zero" << endl;
          throw runtime_error("Executor Error: Division by zero");
          newNode = new BinaryOp(node->getPos()->clone(), l, node->op-
>clone(), r);
          return;
        }
        result = leftVal / rightVal;
      } else {
        newNode = new BinaryOp(node->getPos()->clone(), l, node->op-
>clone(), r);
        return;
      }
      newNode = new IntExp(node->getPos()->clone(), result);
    } else {
      newNode = new BinaryOp(node->getPos()->clone(), l, node->op-
>clone(), r);
    }
  }
```

处理 BinaryOp 节点，遍历其左侧和右侧表达式，并根据操作符执行相应的常量传播逻辑。如果左右两侧都是整数常量，则计算结果并生成一个新的 IntExp 节点；否则，生成一个新的 BinaryOp 节点。

- 修正了一个处理-(1+2)的bug，只需与MinusConverter遇到UnaryOp的实现一致
- 修正了除0的bug，在BinaryOp中增加一个除法的判断，若右值为0，则throw一个error

executor

```
void Executor::visit(MainMethod *node) {
  if (node == nullptr) {
    newNode = nullptr;
    return;
  }
  bool hasReturn = false;
  if (node->sl != nullptr) {
    for (Stm *stm : *(node->sl)) {
      if (stm != nullptr) {
        stm->accept(*this);
        if (dynamic_cast<Return *>(stm) != nullptr) {
          hasReturn = true;
          break; // break when meet the first return statement
        }
      }
    }
  }
  if (!hasReturn) {
    returnValue = 0; // reload returnValue
  }
}
```

在MainMethod中遇到第一个Return就break，防止有多个Return或者Return后面还有语句

```
void Executor::visit(Assign *node) {
  if (node == nullptr) {
    newNode = nullptr;
    return;
  }
  if (node->left != nullptr && node->exp != nullptr) {
    node->exp->accept(*this);
    int value = returnValue;
    IdExp *idExp = dynamic_cast<IdExp *>(node->left);
```

```
      if (idExp != nullptr) {
        variableTable[idExp->id] = value;
      }
    }
  }
```

在Assign时，用一个哈希表存下id对应的value

在BinaryOp中，如果遇到除0，则throw一个runtime_error出来

开发过程