

Cuby

- 课程名称: 编程语言原理与编译
 - 实验项目: 期末大作业
 - 专业班级: 计算机1603
 - 学生学号: 31601147, 31601149
 - 学生姓名: 胡煜, 江瑜
 - 实验指导教师: 郭鸣
 - [github](#)
-

简介

这是一个编译原理大作业, 主要基于microC完成的, 这个之所以取名为Cuby, 主要是在看《[计算的本质](#)》这本书的时候, 发现Ruby是一门非常好玩有趣的语言, 相比C++的错综复杂来说, Ruby是一个集成了优雅与复杂的语言, 比如在`irb`下:

```
>> 3.times{puts("Hello world")}  
Hello world  
Hello world  
Hello world  
=> 3  
>>
```

我看到这门语言的时候我就惊呆了, 居然语言还可以这样玩。而C++却令人反胃, 实在是太恶心了, 虽然说C++给你提供了所有你想用的, 但是学习成本高, 任何东西都感觉不伦不类的。

Ruby是一门完全面向对象的编程语言, 我尝试去实现面向对象的功能, 本来取名叫Yuby, 奈何实现面向对象实在太难了, 我光是看JVM的指令集就很困难了, 还要实现一大堆类库, 实在太过于困难了, 中途也下过Ruby的源代码, 是用C写成的。最后还是放弃了实现面向对象的功能, 选择结合microC与Ruby的语法方面作为我们大作业的方向。

我们打算完善microC并加入Ruby 的 语法, 最后如果还有时间的话能够完成面向对象的一个类(最后还是没时间了)。

结构

- 前端: 由F#语言编写而成
 - `CubyLex.fs1`生成的`CubyLex.fs`词法分析器。
 - `CubyPar.fsy`生成的`CubyPar.fs`语法分析器。
 - `AbstractSyntax.fs` 定义了抽象语法树
 - `Assembly.fs`定义了中间表示的生成指令集
 - `Compile.fs`将抽象语法树转化为中间表示
- 后端: 由Java语言编写而成

- `Machine.java`生成`Machine.class`虚拟机与`Machinetrace.class`堆栈追踪
- 测试集：测试程序放在`testing`文件夹内
- 库：`.net`支持
 - `FsLexYacc.Runtime.dll`

用法

- `fslex --unicode CubyLex.fsl`
生成`CubyLex.fs`词法分析器
- `fsyacc --module CubyPar CubyPar.fsy`
生成`CubyPar.fs`语法分析器与`CubyPar.fsi`
- `javac Machine.java`
生成虚拟机
- `fsi -r FsLexYacc.Runtime.dll AbstractSyntax.fs CubyPar.fs CubyLex.fs Parse.fs Assembly.fs Compile.fs ParseAndComp.fs`
可以启用`fsi`的运行该编译器。
- 在`fsi`中输入:
`open ParseAndComp;;`
- 之后则可以在`fsi`中使用使用：
 - `fromString`：从字符串中进行编译
 - `fromFile`：从文件中进行编译
 - `compileToFile`：生成中间表示

例子：

```
compileToFile (fromFile "testing/ex(struct).c") "testing/ex(struct).out";;  
#q;;  
  
fromString "int a;"
```

生成中间表示之后，便可以使用虚拟机对中间代码进行运行得出结果：

虚拟机功能：

- `java Machine` 运行中间表示
- `java Machinetrace` 追踪堆栈变化

例子：

```
java Machine ex11.out 8
java Machinetrace ex9.out 0
```

功能实现

- 变量定义
 - 简介：原本的microC只有变量声明，我们改进了它使它具有变量定义，且在全局环境与local环境都具有变量定义的功能。
 - 对比

```
// old
int a;
a = 3;
int main(){

    print a;
}
```

```
// new (ex(init).c)
int a = 1;
int b = 2 + 3;

int main(){
    int c = 3;
    print a;
    print b;
    print c;
}
```

- 堆栈图

```
shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby$ java Machine testing/ex\(\init\).out
1 5 3
Ran 0.001 seconds
shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby$ java Machinetrace testing/ex\(\init\).out
[ ]{0: INCSP 1}
[ 0 ]{2: CSTI 0}
[ 0 0 ]{4: CSTI 1}
[ 0 0 1 ]{6: STI}
[ 1 1 ]{7: CSTI 1}
[ 1 1 1 ]{9: CSTI 2}
[ 1 1 1 2 ]{11: CSTI 3}
[ 1 1 1 2 3 ]{13: ADD}
[ 1 1 1 5 ]{14: STI}
[ 1 5 5 ]{15: INCSP -1}
[ 1 5 ]{17: LDARGS}
[ 1 5 ]{18: CALL 0 22}
[ 1 5 21 -999 ]{22: GETBP}
[ 1 5 21 -999 4 ]{23: CSTI 3}
[ 1 5 21 -999 4 3 ]{25: STI}
[ 1 5 21 -999 3 ]{26: CSTI 0}
[ 1 5 21 -999 3 0 ]{28: LDI}
[ 1 5 21 -999 3 1 ]{29: PRINTI}
1 [ 1 5 21 -999 3 1 ]{30: INCSP -1}
[ 1 5 21 -999 3 ]{32: CSTI 1}
[ 1 5 21 -999 3 1 ]{34: LDI}
[ 1 5 21 -999 3 5 ]{35: PRINTI}
5 [ 1 5 21 -999 3 5 ]{36: INCSP -1}
[ 1 5 21 -999 3 ]{38: GETBP}
[ 1 5 21 -999 3 4 ]{39: LDI}
[ 1 5 21 -999 3 3 ]{40: PRINTI}
3 [ 1 5 21 -999 3 3 ]{41: RET 1}
[ 1 5 3 ]{21: STOP}

Ran 0.019 seconds
```

- float 类型

- 简介：浮点型类型，我们将原本的小数转化为二进制表示，再将这二进制转化为整数放入堆栈中，在虚拟机中再转化为小数。
- 例子：
 - 例一：

```
int main(){
    float a = 1.0;
    print a;
}
```

■ 运行栈追踪:

```
shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby/src$ java Machinetrace ../testing/ex\(\float\).out
[0: LDARGS}
[1: CALL 0 5}
4 -999 [5: GETBP}
4 -999 2 [6: CSTF 1065353216}
4 -999 2 1.0 [8: STI}
4 -999 1.0 [9: GETBP}
4 -999 1.0 2 [10: LDI}
4 -999 1.0 1.0 [11: PRINTI}
1.0 [ 4 -999 1.0 1.0 [12: RET 1}
1.0 [4: STOP}

Ran 0.008 seconds
```

• float 类型

- 简介: 浮点型类型, 我们将原本的小数转化为二进制表示
- 例子:

■ 例二:

```
int main(){
    float a = 1.0;
    int b = 2;
    print a+b;
}
```

■ 运行栈追踪:

```
shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby/src$ java Machinetrace ../testing/ex\(\float\).out
[0: LDARGS}
[1: CALL 0 5}
4 -999 [5: GETBP}
4 -999 2 [6: CSTF 1065353216}
4 -999 2 1.0 [8: STI}
4 -999 1.0 [9: GETBP}
4 -999 1.0 2 [10: CSTI 1}
4 -999 1.0 2 1 [12: ADD}
4 -999 1.0 3 [13: CSTI 2}
4 -999 1.0 3 2 [15: STI}
4 -999 1.0 2 [16: GETBP}
4 -999 1.0 2 2 [17: LDI}
4 -999 1.0 2 1.0 [18: GETBP}
4 -999 1.0 2 1.0 2 [19: CSTI 1}
4 -999 1.0 2 1.0 2 1 [21: ADD}
4 -999 1.0 2 1.0 3 [22: LDI}
4 -999 1.0 2 1.0 2 [23: ADD}
4 -999 1.0 2 3.0 [24: PRINTI}
3.0 [ 4 -999 1.0 2 3.0 [25: RET 2}
3.0 [4: STOP}

Ran 0.017 seconds
```

• 例二:

```
int main(){
    float a = 1.0;
    int b = 2;
    print a+b;
}
```

• float 类型

- 简介:

• char 类型

- 简介: 原本电脑
- 例子:

```
int main ()
{
    char c = 'c';
    print c;
}
```

- 运行栈追踪:

```
shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby/src$ java Machinetrace ../testing/ex\chars\).out
[ ]{0: LDARGS}
[ ]{1: CALL 0 5}
[ 4 -999 ]{5: GETBP}
[ 4 -999 2 ]{6: <unknown>}
[ 4 -999 2 c ]{8: STI} lename) throws FileNotFoundException, IOExcepti
[ 4 -999 c ]{9: GETBP}
[ 4 -999 c 2 ]{10: LDI} c>());
[ 4 -999 c c ]{11: PRINTI}
c [ 4 -999 c c ]{12: RET 1}
[ c ]{4: STOP}

Ran 0.006 seconds
```

- 自增操作
 - 简介:包含i++ ++i 操作
 - 例子:

```
int main(){
    int n;
    int a;
    n = 2;
    a = ++n;
    a = n++;
}
```

- 运行栈追踪:

```
E:\学习\编译语言\Yuby\Yuby\Cuby>java Machinetrace testing/ex(selfplus).out
[ ] {0: LDARGS}
[ ] {1: CALL 0 5}
[ 4 -999 ] {5: INCSP 1}
[ 4 -999 0 ] {7: INCSP 1}
[ 4 -999 0 0 ] {9: GETBP}
[ 4 -999 0 0 2 ] {10: CSTI 2}
[ 4 -999 0 0 2 2 ] {12: STI}
[ 4 -999 2 0 2 ] {13: INCSP -1}
[ 4 -999 2 0 ] {15: GETBP}
[ 4 -999 2 0 2 ] {16: CSTI 1}
[ 4 -999 2 0 2 1 ] {18: ADD}
[ 4 -999 2 0 3 ] {19: GETBP}
[ 4 -999 2 0 3 2 ] {20: LDI}
[ 4 -999 2 0 3 2 ] {21: CSTI 1}
[ 4 -999 2 0 3 2 1 ] {23: ADD}
[ 4 -999 2 0 3 3 ] {24: INCSP -1}
[ 4 -999 2 0 3 ] {26: GETBP}
[ 4 -999 2 0 3 2 ] {27: GETBP}
[ 4 -999 2 0 3 2 2 ] {28: LDI}
[ 4 -999 2 0 3 2 2 ] {29: CSTI 1}
[ 4 -999 2 0 3 2 2 1 ] {31: ADD}
[ 4 -999 2 0 3 2 3 ] {32: STI}
[ 4 -999 3 0 3 3 ] {33: STI}
[ 4 -999 3 3 3 ] {34: INCSP -1}
[ 4 -999 3 3 ] {36: GETBP}
[ 4 -999 3 3 2 ] {37: CSTI 1}
[ 4 -999 3 3 2 1 ] {39: ADD}
[ 4 -999 3 3 3 ] {40: GETBP}
[ 4 -999 3 3 3 2 ] {41: LDI}
[ 4 -999 3 3 3 3 ] {42: GETBP}
[ 4 -999 3 3 3 3 2 ] {43: GETBP}
[ 4 -999 3 3 3 3 2 2 ] {44: LDI}
[ 4 -999 3 3 3 3 2 3 ] {45: CSTI 1}
[ 4 -999 3 3 3 3 2 3 1 ] {47: ADD}
[ 4 -999 3 3 3 3 2 4 ] {48: STI}
[ 4 -999 4 3 3 3 4 ] {49: INCSP -1}
[ 4 -999 4 3 3 3 ] {51: STI}
[ 4 -999 4 3 3 ] {52: RET 2}
[ 3 ] {4: STOP}
```

- FOR循环

- 简介: 增加了for循环, 以及类似于Ruby的循环
- 例子:

```
int main(){
    int i;
    i = 0;
    int n;
    n = 0;
    for(i = 0 ; i < 5 ; ++i){
        n = n + i;
    }
}
```

- 运行栈追踪:

```
[ 4 -999 3 3 ] {28: GETBP}
[ 4 -999 3 3 2 ] {29: CSTI 1}
```

```
[ 4 -999 3 3 2 1 ]{29: CSTI 1}
[ 4 -999 3 3 2 1 ]{31: ADD}
[ 4 -999 3 3 3 1 ]{32: GETBP}
[ 4 -999 3 3 3 2 ]{33: CSTI 1}
[ 4 -999 3 3 3 2 1 ]{35: ADD}
[ 4 -999 3 3 3 3 1 ]{36: LDI}
[ 4 -999 3 3 3 3 1 ]{37: GETBP}
[ 4 -999 3 3 3 3 2 ]{38: LDI}
[ 4 -999 3 3 3 3 3 ]{39: ADD}
[ 4 -999 3 3 3 6 1 ]{40: STI}
[ 4 -999 3 6 6 1 ]{41: INCSP -1}
[ 4 -999 3 6 1 ]{43: GETBP}
[ 4 -999 3 6 2 1 ]{44: LDI}
[ 4 -999 3 6 3 1 ]{45: CSTI 1}
[ 4 -999 3 6 3 1 ]{47: ADD}
[ 4 -999 3 6 4 1 ]{48: INCSP -1}
[ 4 -999 3 6 1 ]{50: GETBP}
[ 4 -999 3 6 2 1 ]{51: GETBP}
[ 4 -999 3 6 2 2 1 ]{52: LDI}
[ 4 -999 3 6 2 3 1 ]{53: CSTI 1}
[ 4 -999 3 6 2 3 1 ]{55: ADD}
[ 4 -999 3 6 2 4 1 ]{56: STI}
[ 4 -999 4 6 4 1 ]{57: INCSP -1}
[ 4 -999 4 6 1 ]{59: GETBP}
[ 4 -999 4 6 2 1 ]{60: LDI}
[ 4 -999 4 6 4 1 ]{61: CSTI 5}
[ 4 -999 4 6 4 5 1 ]{63: LT}
[ 4 -999 4 6 1 1 ]{64: IFNZRO 28}
[ 4 -999 4 6 1 ]{28: GETBP}
[ 4 -999 4 6 2 1 ]{29: CSTI 1}
[ 4 -999 4 6 2 1 1 ]{31: ADD}
[ 4 -999 4 6 3 1 ]{32: GETBP}
[ 4 -999 4 6 3 2 1 ]{33: CSTI 1}
[ 4 -999 4 6 3 2 1 1 ]{35: ADD}
[ 4 -999 4 6 3 3 1 ]{36: LDI}
[ 4 -999 4 6 3 6 1 ]{37: GETBP}
[ 4 -999 4 6 3 6 2 1 ]{38: LDI}
[ 4 -999 4 6 3 6 4 1 ]{39: ADD}
[ 4 -999 4 6 3 10 1 ]{40: STI}
[ 4 -999 4 10 10 1 ]{41: INCSP -1}
[ 4 -999 4 10 1 ]{43: GETBP}
[ 4 -999 4 10 2 1 ]{44: LDI}
[ 4 -999 4 10 4 1 ]{45: CSTI 1}
[ 4 -999 4 10 4 1 1 ]{47: ADD}
[ 4 -999 4 10 5 1 ]{48: INCSP -1}
[ 4 -999 4 10 1 ]{50: GETBP}
[ 4 -999 4 10 2 1 ]{51: GETBP}
[ 4 -999 4 10 2 2 1 ]{52: LDI}
[ 4 -999 4 10 2 4 1 ]{53: CSTI 1}
[ 4 -999 4 10 2 4 1 1 ]{55: ADD}
[ 4 -999 4 10 2 5 1 ]{56: STI}
[ 4 -999 5 10 5 1 ]{57: INCSP -1}
```



```
[ 4 -999 5 10 ] {59: GETBP}
[ 4 -999 5 10 2 ] {60: LDI}
[ 4 -999 5 10 5 ] {61: CSTI 5}
[ 4 -999 5 10 5 5 ] {63: LT}
[ 4 -999 5 10 0 ] {64: IFNZRO 28}
```

```
int main()
{
    int n;
    int s;
    s = 0;
    for n in (3..7)
    {
        s = s+n;
    }
}
```

o 运行栈追踪:

```
E:\学习\编译语言\Yuby\Yuby\Cuby>java Machinetrace testing/ex(range).out
[ ] {0: LDARGS}
[ ] {1: CALL 0 5}
[ 4 -999 ] {5: INCSP 1}
[ 4 -999 0 ] {7: INCSP 1}
[ 4 -999 0 0 ] {9: GETBP}
[ 4 -999 0 0 2 ] {10: CSTI 1}
[ 4 -999 0 0 2 1 ] {12: ADD}
[ 4 -999 0 0 3 ] {13: CSTI 0}
[ 4 -999 0 0 3 0 ] {15: STI}
[ 4 -999 0 0 0 ] {16: INCSP -1}
[ 4 -999 0 0 ] {18: GETBP}
[ 4 -999 0 0 2 ] {19: CSTI 3}
[ 4 -999 0 0 2 3 ] {21: STI}
[ 4 -999 3 0 3 ] {22: INCSP -1}
[ 4 -999 3 0 ] {24: GOTO 50}
[ 4 -999 3 0 ] {50: GETBP}
[ 4 -999 3 0 2 ] {51: LDI}
[ 4 -999 3 0 3 ] {52: CSTI 7}
[ 4 -999 3 0 3 7 ] {54: LT}
[ 4 -999 3 0 1 ] {55: IFNZRO 26}
[ 4 -999 3 0 ] {26: GETBP}
[ 4 -999 3 0 2 ] {27: CSTI 1}
[ 4 -999 3 0 2 1 ] {29: ADD}
[ 4 -999 3 0 3 ] {30: GETBP}
[ 4 -999 3 0 3 2 ] {31: CSTI 1}
[ 4 -999 3 0 3 2 1 ] {33: ADD}
[ 4 -999 3 0 3 3 ] {34: LDI}
[ 4 -999 3 0 3 0 ] {35: GETBP}
[ 4 -999 3 0 3 0 2 ] {36: LDI}
[ 4 -999 3 0 3 0 3 ] {37: ADD}
[ 4 -999 3 0 3 3 ] {38: STI}
[ 4 -999 3 3 3 ] {39: INCSP -1}
[ 4 -999 3 3 ] {41: GETBP}
[ 4 -999 3 3 2 ] {42: GETBP}
```

- 三目运算符
 - 简介：三目运算符 $a > b ? a : b$
 - 用例：

```
int main()
{
    int a=0;
    int b=7;
    int c = a>b?a:b;
}
```

- 运行栈追踪：

```
E:\学习\编译语言\Yuby\Yuby\Cuby>java Machinetrace testing/ex(ternary).out
[ ] {0: LDARGS}
[ ] {1: CALL 0 5}
[ 4 -999 ] {5: GETBP}
[ 4 -999 2 ] {6: CSTI 0}
[ 4 -999 2 0 ] {8: STI}
[ 4 -999 0 ] {9: GETBP}
[ 4 -999 0 2 ] {10: CSTI 1}
[ 4 -999 0 2 1 ] {12: ADD}
[ 4 -999 0 3 ] {13: CSTI 7}
[ 4 -999 0 3 7 ] {15: STI}
[ 4 -999 0 7 ] {16: GETBP}
[ 4 -999 0 7 2 ] {17: CSTI 2}
[ 4 -999 0 7 2 2 ] {19: ADD}
[ 4 -999 0 7 4 ] {20: GETBP}
[ 4 -999 0 7 4 2 ] {21: LDI}
[ 4 -999 0 7 4 0 ] {22: GETBP}
[ 4 -999 0 7 4 0 2 ] {23: CSTI 1}
[ 4 -999 0 7 4 0 2 1 ] {25: ADD}
[ 4 -999 0 7 4 0 3 ] {26: LDI}
[ 4 -999 0 7 4 0 7 ] {27: SWAP}
[ 4 -999 0 7 4 7 0 ] {28: LT}
[ 4 -999 0 7 4 0 ] {29: IFZERO 35}
[ 4 -999 0 7 4 ] {35: GETBP}
[ 4 -999 0 7 4 2 ] {36: CSTI 1}
[ 4 -999 0 7 4 2 1 ] {38: ADD}
[ 4 -999 0 7 4 3 ] {39: LDI}
[ 4 -999 0 7 4 7 ] {40: STI}
[ 4 -999 0 7 7 ] {41: RET 2}
[ 7 ] {4: STOP}

Ran 0.053 seconds
```

- do - while
 - 简介：在判断前先运行body中的操作。
 - 例子：

```
int main()
{
    int n=2;
    do{
        n++;
    }
```

```
    }while(n<0);
}
```

◦ 运行栈追踪:

- n++被执行
- n的终值为3 处于栈中2的位置
- 堆栈图:

```
E:\学习\编译语言\Yuby\Yuby\Cuby>java Machinetrace testing/ex(dowhile).out
[ ] {0: LDARGS}
[ ] {1: CALL 0 5}
[ 4 -999 ] {5: GETBP}
[ 4 -999 2 ] {6: CSTI 2}
[ 4 -999 2 2 ] {8: STI}
[ 4 -999 2 ] {9: GETBP}
[ 4 -999 2 2 ] {10: LDI}
[ 4 -999 2 2 ] {11: GETBP}
[ 4 -999 2 2 2 ] {12: GETBP}
[ 4 -999 2 2 2 2 ] {13: LDI}
[ 4 -999 2 2 2 2 ] {14: CSTI 1}
[ 4 -999 2 2 2 2 1 ] {16: ADD}
[ 4 -999 2 2 2 3 ] {17: STI}
[ 4 -999 3 2 3 ] {18: INCSP -2}
[ 4 -999 3 ] {20: GETBP}
[ 4 -999 3 2 ] {21: LDI}
[ 4 -999 3 3 ] {22: CSTI 0}
[ 4 -999 3 3 0 ] {24: LT}
[ 4 -999 3 0 ] {25: IFNZRO 9}
[ 4 -999 3 ] {27: RET 0}
[ 3 ] {4: STOP}

Ran 0.032 seconds
```

• 类似C的switch-case

- 当没有break时，匹配到一个case后，会往下执行所以case的body
- 若当前没有匹配的case时，不会执行body，会一直往下找匹配的case
- 之前的实现是递归匹配每个case，当前类似C语言的switch-case实现上在label的设立更为复杂一些。
- 例子:

```
int main(){
    int i=0;
    int n=1;
    switch(n){
        case 1:i=n+n;
        case 5:i=i+n*n;
    }
}
```

◦ 运行栈追踪:

- n的值与case1 匹配，没有break， $i=n+n$ 与case 5 中的 $i+n*n$ 都被执行
- i的结果为 $(1+1) + 1*1 = 3$
- 栈中3的位置为i， 4的位置为n
- 堆栈图：

```
[ 4 -999 0 1 ] {26: GETBP}
[ 4 -999 0 1 2 ] {27: GETBP}
[ 4 -999 0 1 2 2 ] {28: CSTI 1}
[ 4 -999 0 1 2 2 1 ] {30: ADD}
[ 4 -999 0 1 2 3 ] {31: LDI}
[ 4 -999 0 1 2 1 ] {32: GETBP}
[ 4 -999 0 1 2 1 2 ] {33: CSTI 1}
[ 4 -999 0 1 2 1 2 1 ] {35: ADD}
[ 4 -999 0 1 2 1 3 ] {36: LDI}
[ 4 -999 0 1 2 1 1 ] {37: ADD}
[ 4 -999 0 1 2 2 ] {38: STI}
[ 4 -999 2 1 2 ] {39: INCSP -1}
[ 4 -999 2 1 ] {41: GOTO 53}
[ 4 -999 2 1 ] {53: GETBP}
[ 4 -999 2 1 2 ] {54: GETBP}
[ 4 -999 2 1 2 2 ] {55: LDI}
[ 4 -999 2 1 2 2 ] {56: GETBP}
[ 4 -999 2 1 2 2 2 ] {57: CSTI 1}
[ 4 -999 2 1 2 2 2 1 ] {59: ADD}
[ 4 -999 2 1 2 2 3 ] {60: LDI}
[ 4 -999 2 1 2 2 1 ] {61: GETBP}
[ 4 -999 2 1 2 2 1 2 ] {62: CSTI 1}
[ 4 -999 2 1 2 2 1 2 1 ] {64: ADD}
[ 4 -999 2 1 2 2 1 3 ] {65: LDI}
[ 4 -999 2 1 2 2 1 1 ] {66: MUL}
[ 4 -999 2 1 2 2 1 ] {67: ADD}
[ 4 -999 2 1 2 3 ] {68: STI}
[ 4 -999 3 1 3 ] {69: RET 2}
[ 3 ] {4: STOP}

Ran 0.053 seconds
```

- break功能
 - 在for while switch 中，都加入break功能
 - 维护Label表来实现
 - 例子：与没有break的switch进行对比：

```
int main(){
    int i=0;
    int n=1;
    switch(n){
        case 1:{i=n+n;break;}
        case 5:i=i+n*n;
    }
}
```

- 运行栈追踪

- n的值与case1 匹配，执行i=n+n，遇到break结束。
- i的结果为(1+1)=2
- 栈中3的位置为i，4的位置为n
- 堆栈图：

```
E:\学习\编译语言\Yuby\Yuby\Cuby>java Machinetrace testing/ex(break).out
[ ] {0: LDARGS}
[ ] {1: CALL 0 5}
[ 4 -999 ] {5: GETBP}
[ 4 -999 2 ] {6: CSTI 0}
[ 4 -999 2 0 ] {8: STI}
[ 4 -999 0 ] {9: GETBP}
[ 4 -999 0 2 ] {10: CSTI 1}
[ 4 -999 0 2 1 ] {12: ADD}
[ 4 -999 0 3 ] {13: CSTI 1}
[ 4 -999 0 3 1 ] {15: STI}
[ 4 -999 0 1 ] {16: GETBP}
[ 4 -999 0 1 2 ] {17: CSTI 1}
[ 4 -999 0 1 2 1 ] {19: ADD}
[ 4 -999 0 1 3 ] {20: LDI}
[ 4 -999 0 1 1 ] {21: CSTI 1}
[ 4 -999 0 1 1 1 ] {23: EQ}
[ 4 -999 0 1 1 ] {24: IFZERO 43}
[ 4 -999 0 1 ] {26: GETBP}
[ 4 -999 0 1 2 ] {27: GETBP}
[ 4 -999 0 1 2 2 ] {28: CSTI 1}
[ 4 -999 0 1 2 2 1 ] {30: ADD}
[ 4 -999 0 1 2 3 ] {31: LDI}
[ 4 -999 0 1 2 1 ] {32: GETBP}
[ 4 -999 0 1 2 1 2 ] {33: CSTI 1}
[ 4 -999 0 1 2 1 2 1 ] {35: ADD}
[ 4 -999 0 1 2 1 3 ] {36: LDI}
[ 4 -999 0 1 2 1 1 ] {37: ADD}
[ 4 -999 0 1 2 2 ] {38: STI}
[ 4 -999 2 1 2 ] {39: INCSP -1}
[ 4 -999 2 1 ] {41: GOTO 71}
[ 4 -999 2 1 ] {71: RET 1}
```

- continue 功能

- 在for while 中加入continue功能
- 例子：

```
int main()
{
    int i ;
    int n = 0;
    for(i=0;i<5;i++)
    {
        if(i<2)
            continue;
        if(i>3)
            break;
        n=n+i;
    }
}
```

- 运行栈追踪：
 - `i=0 1` 的时候`continue` `i>3` 的时候`break`
 - `n = 2 + 3` 结果为5
 - 栈中3的位置为i, 4的位置为n
 - 堆栈图:

```
[ 4 -999 3 2 3 2 2 ] {51: LDI}
[ 4 -999 3 2 3 2 3 ] {52: ADD}
[ 4 -999 3 2 3 5 ] {53: STI}
[ 4 -999 3 5 5 ] {54: INCSP -1}
[ 4 -999 3 5 ] {56: GETBP}
[ 4 -999 3 5 2 ] {57: LDI}
[ 4 -999 3 5 3 ] {58: GETBP}
[ 4 -999 3 5 3 2 ] {59: GETBP}
[ 4 -999 3 5 3 2 2 ] {60: LDI}
[ 4 -999 3 5 3 2 3 ] {61: CSTI 1}
[ 4 -999 3 5 3 2 3 1 ] {63: ADD}
[ 4 -999 3 5 3 2 4 ] {64: STI}
[ 4 -999 4 5 3 4 ] {65: INCSP -2}
[ 4 -999 4 5 ] {67: GETBP}
[ 4 -999 4 5 2 ] {68: LDI}
[ 4 -999 4 5 4 ] {69: CSTI 5}
[ 4 -999 4 5 4 5 ] {71: LT}
[ 4 -999 4 5 1 ] {72: IFNZRO 22}
[ 4 -999 4 5 ] {22: GETBP}
[ 4 -999 4 5 2 ] {23: LDI}
[ 4 -999 4 5 4 ] {24: CSTI 2}
[ 4 -999 4 5 4 2 ] {26: LT}
[ 4 -999 4 5 0 ] {27: IFZERO 31}
[ 4 -999 4 5 ] {31: GETBP}
[ 4 -999 4 5 2 ] {32: LDI}
[ 4 -999 4 5 4 ] {33: CSTI 3}
[ 4 -999 4 5 4 3 ] {35: SWAP}
[ 4 -999 4 5 3 4 ] {36: LT}
[ 4 -999 4 5 1 ] {37: IFZERO 41}
[ 4 -999 4 5 ] {39: GOTO 74}
[ 4 -999 4 5 ] {74: RET 1}
[ 5 ] {4: STOP}
```

- 结构体功能:

- 简介: 加入了C中的结构体功能, 可以嵌套数组
- 首先, 先创建结构体定义表, 用来查找, 结构体定义表中包含结构体的总体大小, 名字, 以及变量和偏移量。然后查找结构体变量表, 加入该变量到`varEnv`中, 访问成员时, 便可以通过`.`运算符, 通过偏移值转化为简单指令集。
- 例子:

```

struct student{
    int number;
    int number2;
    char name[5];
    float id;
};

int main(){
    struct student hello;
    hello.number = 10;
    hello.id = 234;
    hello.name[4] = 'c';
    hello.name[0] = 'a';
    print hello.number;
    print hello.name[4];
}

```

■ 运行栈追踪

```

shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby/src$ java Machinetrace ../testing/ex\(\struct\).out
[ ]{0: LDARGS}
[ ]{1: CALL 0 5}
[ 4 -999 ]{5: INCSP 8}
[ 4 -999 null null null null null null null null ]{7: GETSP}
[ 4 -999 null null null null null null null null 9 ]{8: CSTI 7}
[ 4 -999 null null null null null null null null 9 7 ]{10: SUB}
[ 4 -999 null null null null null null null null 2 ]{11: GETBP}
[ 4 -999 null null null null null null null null 2 2 ]{12: CSTI 10}
[ 4 -999 null null null null null null null null 2 2 10 ]{14: STI}
[ 4 -999 10 null null null null null null null 2 10 ]{15: INCSP -1}
[ 4 -999 10 null null null null null null null 2 ]{17: GETBP}
[ 4 -999 10 null null null null null null null 2 2 ]{18: CSTI 7}
[ 4 -999 10 null null null null null null null 2 2 7 ]{20: ADD}
[ 4 -999 10 null null null null null null null 2 9 ]{21: CSTI 234}
[ 4 -999 10 null null null null null null null 2 9 234 ]{23: STI}
[ 4 -999 10 null null null null null null null 234 2 234 ]{24: INCSP -1}
[ 4 -999 10 null null null null null null null 234 2 ]{26: GETBP}
[ 4 -999 10 null null null null null null null 234 2 2 ]{27: CSTI 2}
[ 4 -999 10 null null null null null null null 234 2 2 2 ]{29: ADD}
[ 4 -999 10 null null null null null null null 234 2 4 ]{30: CSTI 4}
[ 4 -999 10 null null null null null null null 234 2 4 4 ]{32: ADD}
[ 4 -999 10 null null null null null null null 234 2 8 ]{33: CSTC c}
[ 4 -999 10 null null null null null null null 234 2 8 c ]{35: STI}
[ 4 -999 10 null null null null null null c 234 2 c ]{36: INCSP -1}
[ 4 -999 10 null null null null null null c 234 2 ]{38: GETBP}
[ 4 -999 10 null null null null null null c 234 2 2 ]{39: CSTI 2}
[ 4 -999 10 null null null null null null c 234 2 2 2 ]{41: ADD}
[ 4 -999 10 null null null null null null c 234 2 4 ]{42: CSTC a}
[ 4 -999 10 null null null null null null c 234 2 4 a ]{44: STI}
[ 4 -999 10 null a null null null null c 234 2 a ]{45: INCSP -1}
[ 4 -999 10 null a null null null null c 234 2 ]{47: GETBP}
[ 4 -999 10 null a null null null null c 234 2 2 ]{48: LDI}
[ 4 -999 10 null a null null null null c 234 2 10 ]{49: PRINTI}
10 [ 4 -999 10 null a null null null c 234 2 10 ]{50: INCSP -1}
[ 4 -999 10 null a null null null c 234 2 ]{52: GETBP}
[ 4 -999 10 null a null null null c 234 2 2 ]{53: CSTI 2}
[ 4 -999 10 null a null null null c 234 2 2 2 ]{55: ADD}
[ 4 -999 10 null a null null null c 234 2 4 ]{56: CSTI 4}
[ 4 -999 10 null a null null null c 234 2 4 4 ]{58: ADD}
[ 4 -999 10 null a null null null c 234 2 8 ]{59: LDI}
[ 4 -999 10 null a null null null c 234 2 c ]{60: PRINTI}
c [ 4 -999 10 null a null null null c 234 2 c ]{61: RET 9}
[ c ]{4: STOP}

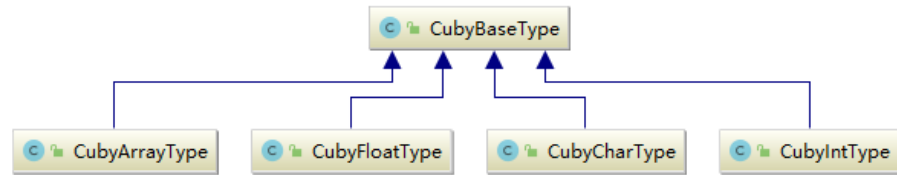
```

Ran 0.044 seconds

■ shiro@DESKTOP-6IOT538:/mnt/d/Yuby/Cuby/src\$

- JVM
 - 简介:

- 将之前的虚拟机重新写了一边，原先的虚拟机只能针对int类型进行存储和操作。我具体定义了一些类型，使虚拟机具有较强的拓展性。
- 类型继承关系图：



-
- 指令集添加：
 - CSTF:
 - 简介: const float
 - 功能: 在堆栈中加入一个float
 - CSTC:
 - 简介: const char
 - 功能: 在堆栈中加入一个char
- 运行时异常：
 - OperatorError:
 - 简介: 在计算时发生的异常，例如除数为0时，出现异常。
 - ImcompatibleTypeError:
 - 简介: 类型不匹配的时候进行时出现的异常，例如'a' + 1.0 抛出异常。

- try-catch:

- 简介：
 - 除0异常捕获：
 - 虚拟机运行时的除0异常
 - 出现显式的除0时THROW 异常
 - 寄存器添加：
 - hr: 保存当前异常在栈中的地址，用于追踪需要处理的异常
 - 指令添加：
 - PUSHHDLR,保存catch中的异常种类，需要跳转的位置以及hr入栈
 - POPHDLR，与PUSHHDLR对应
 - THROW，用于丢出异常，从hr开始找匹配
 - 例子：
- 目前做了有关除0异常的内容，由于没有异常类，暂且通过字符串的方式作为异常种类，将异常编号匹配。解决了运行时try-catch变量环境的问题，解决了异常处理时栈环境变化带来的空间影响。能够正常的匹配到异常。

```

int main()
{
    try{
        int a=0;
        int n=5;
    }
}
  
```



```

        n=n/a;
    }
    catch("ArithmeticalExcpption")
    {
        n=0;
        print n;
    }
}

```

■ 运行时堆栈:

```

E:\学习\编译语言\Yuby\Yuby\Cuby\src>java Machinetrace ../testing/ex(exception).out
[ ] {0: LDARGS}
[ ] {1: CALL 0 5}
[ 4 -999 ] {5: PUSHHR 1 41}
[ 4 -999 1 41 -1 ] {8: GETBP}
[ 4 -999 1 41 -1 2 ] {9: CSTI 3}
[ 4 -999 1 41 -1 2 3 ] {11: ADD}
[ 4 -999 1 41 -1 5 ] {12: CSTI 0}
[ 4 -999 1 41 -1 5 0 ] {14: STI}
[ 4 -999 1 41 -1 0 ] {15: GETBP}
[ 4 -999 1 41 -1 0 2 ] {16: CSTI 4}
[ 4 -999 1 41 -1 0 2 4 ] {18: ADD}
[ 4 -999 1 41 -1 0 6 ] {19: CSTI 5}
[ 4 -999 1 41 -1 0 6 5 ] {21: STI}
[ 4 -999 1 41 -1 0 5 ] {22: GETBP}
[ 4 -999 1 41 -1 0 5 2 ] {23: CSTI 4}
[ 4 -999 1 41 -1 0 5 2 4 ] {25: ADD}
[ 4 -999 1 41 -1 0 5 6 ] {26: GETBP}
[ 4 -999 1 41 -1 0 5 6 2 ] {27: CSTI 4}
[ 4 -999 1 41 -1 0 5 6 2 4 ] {29: ADD}
[ 4 -999 1 41 -1 0 5 6 6 ] {30: LDI}
[ 4 -999 1 41 -1 0 5 6 5 ] {31: GETBP}
[ 4 -999 1 41 -1 0 5 6 5 2 ] {32: CSTI 3}
[ 4 -999 1 41 -1 0 5 6 5 2 3 ] {34: ADD}
[ 4 -999 1 41 -1 0 5 6 5 5 ] {35: LDI}
[ 4 -999 1 41 -1 0 5 6 5 0 ] {36: DIV}
hr:2 exception:1
[ 4 -999 ] {41: GETBP}
[ 4 -999 2 ] {42: CSTI 4}
[ 4 -999 2 4 ] {44: ADD}
[ 4 -999 6 ] {45: CSTI 0}
[ 4 -999 6 0 ] {47: STI}
[ 4 -999 0 ] {48: INCSP -1}
[ 4 -999 ] {50: GETBP}
[ 4 -999 2 ] {51: CSTI 4}
[ 4 -999 2 4 ] {53: ADD}
[ 4 -999 6 ] {54: LDI}
[ 4 -999 0 ] {55: PRINTI}
0 [ 4 -999 0 ] {56: INCSP -1}
[ 4 -999 ] {58: RET -1}
• [ -999 ] {4: STOP}

```

心得体会

- 胡煜:

这学期大作业比较多,中间的时候也写了一点编译原理,不过看见F#就头疼,不仅是因为它比较新,是一个完全没接触过的全新类型的语言,同时还是因为它的资料实在太少了,一切都得自己慢慢摸索。实在是痛苦。而且这个作业嘛,写了一点,因为其他事情耽搁了几天,而且隔了两天就差不多忘了。又得

重拾起来学习，实在痛苦。其实老师上课的内容，其实每节课都是感觉学得一知半解，东西实在太多，也比较难以消化，隔了几天就全忘得差不多了。不过通过这次大作业之后，收获颇丰。

- 了解了函数式编程的语言特点与特性，拓宽视野。
- 利用F#与java完善了一个编程语言的从前端到后端的完整的搭建
- 清楚了一些关于C语言的设计方法与局限性。
- 理解了栈式虚拟机的工作原理与一些设计方法
- 利用虚拟机避免与汇编指令集直接打交道，优化代码执行策略

时间太紧了，如果时间再多一点的话，还可以往虚拟机中加入全局静态变量表、完善结构体、加入头文件机制、加入输入模块等。

总而言之，这节编译原理课，收获颇丰，F#好用值得学习。编译原理的世界还是相当有趣的。

- 江瑜：

本学期的编译原理学习，还是非常需要时间的。无论是前期各星期的作业以及大作业，都需要投入一定的精力去了解，去学习在编译之中理论以及实践的内容。编译原理的基本理念以及后期做一个“玩具型”语言的过程，感觉都对计算机学习有着极大的帮助。介于机器与程序之间的桥梁，让我更深一步了解一些语言模型，机器模型，语义语法结构等。郭老师对于，道，法，术三者在此方面的理念，我觉得很有道理，确实，大学就应该加深对于计算本质的理解与运用。总结一下

- 大作业完成中，对指令与栈结构有了更深的了解
- 随着对fsharp的使用越加频繁，也加深了对函数式编程语言的印象
- 过程的积累中，逐渐了解一些编译的理念，计算的思维。
- 在大作业有些功能的完成上，还可以再优化精简。时间足够的话还要继续完善更多的异常处理功能，可以增加一些面向对象的思想在里面，对虚拟机的支持要求更高，也可以做些关于垃圾回收的功能。

技术评价

功能	对应文件	优	良	中
变量声明定义	ex(init).c	√		
自增、自减	ex(selfplus).c	√		
for循环	ex(for).c	√		
三目运算符	ex(ternary).c	√		
do-while	ex(dowhile).c	√		
while	ex(while).c	√		
range循环	ex(range).c		√	
break	ex(break).c	√		
continue	ex(continue).c	√		
switch-case	ex(switch).c	√		
float 类型	ex(float).c	√		
char 类型	ex(chars).c	√		

功能	对应文件	优	良	中
struct结构体	ex(struct).c	√		
try-catch	ex(exception).c	√		
虚拟机类型支持	Machine.java	√		
虚拟机异常	exception	√		

小组分工

- 胡煜
 - 学号：31601147
 - 班级：计算机1603
 - 工作内容
 - 文档编写
 - 测试程序
 - 主要负责虚拟机和中间代码生成
 - 结构体
- 江瑜
 - 学号：31601149
 - 班级：计算机1603
 - 工作内容
 - 文档编写
 - 测试程序
 - 语法分析
 - 词法分析
 - 栈、堆设计
 - try-catch
- 权重分配表：

胡煜	江瑜
0.95	0.95