testcase1

```
int x = 1;
bool b(bool x){
    return x;
}
class C{
    C(){
        x();
        {int x;}
    }
    void x(){}
}
int main(){
    string x = "789dcbadcba\n";
    int y;
    C c;
    c.x();
    b(true);
    {
        y = x.parseInt();
        int x = y;
     }
    return y;
}
```

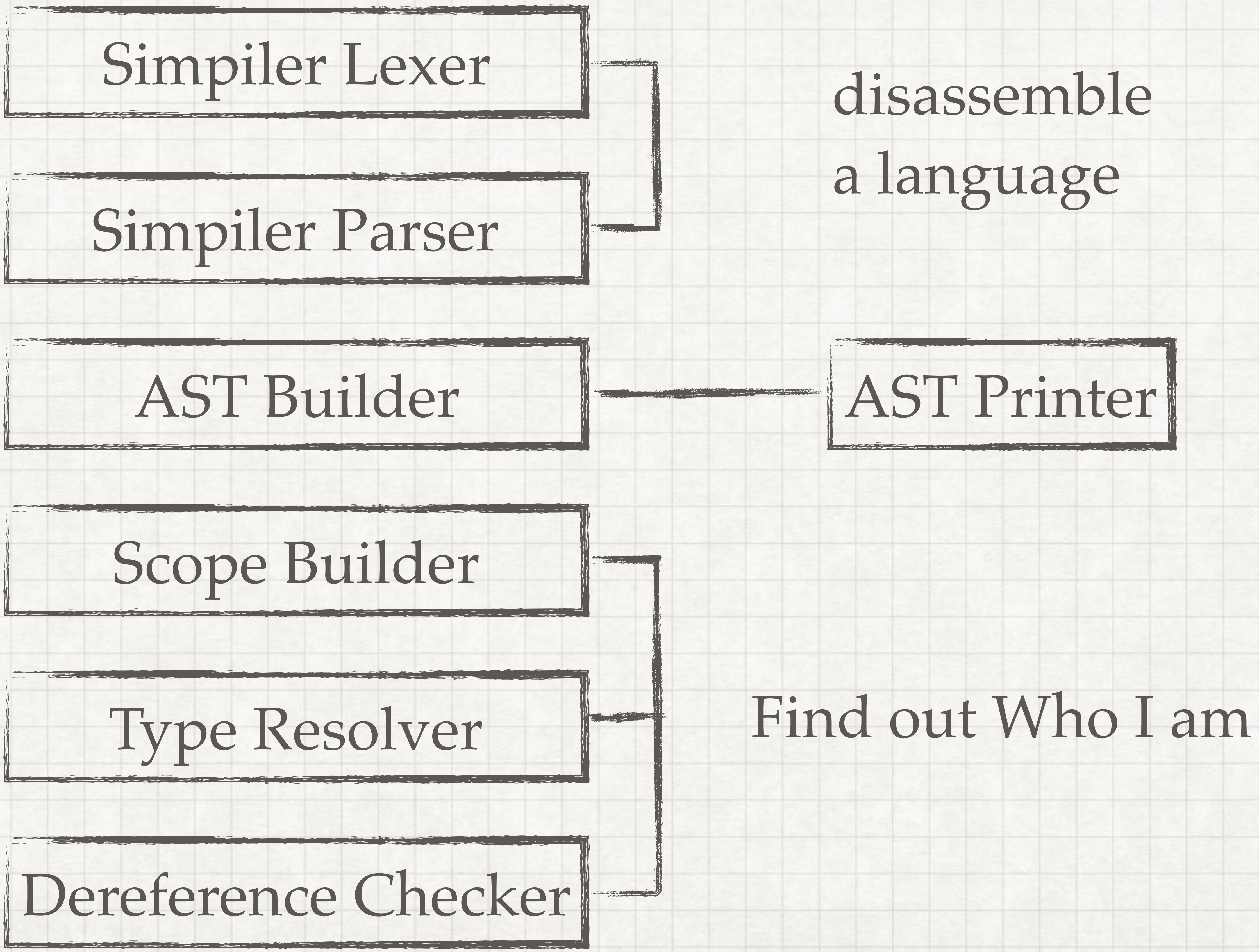
testcase2

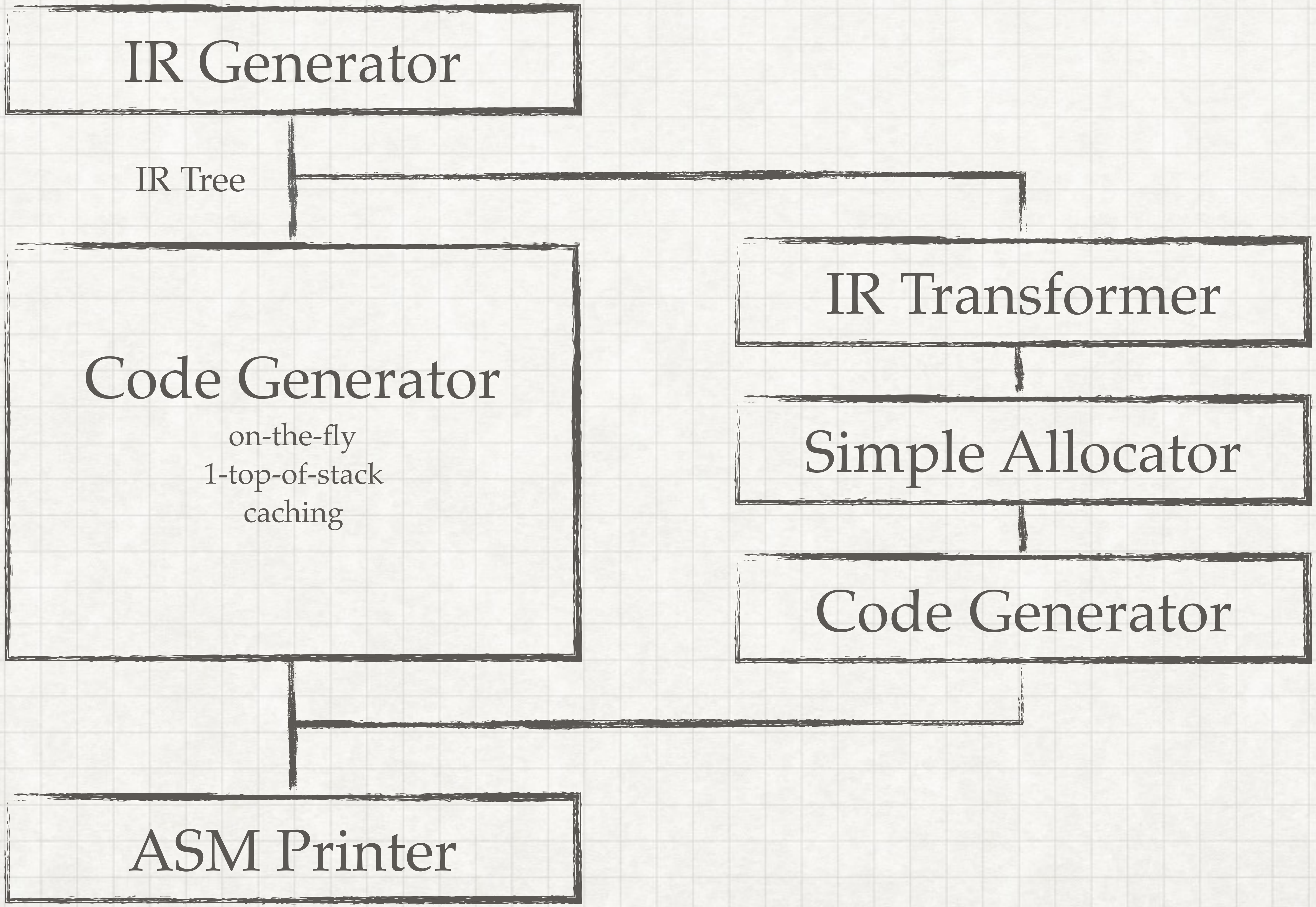
```
class Edge{
    int to;
    int next;
}

Edge[] e;
int ne = 0;
int n = 0;
int[] head;
int ans = 0;
int size = 0;
int i = 0;

void add(int from, int to){
    ++ne;
    e[ne].to = to;
    e[ne].next = head[from];
    head[from] = ne;
}

bool[] visit;
int[] son;
int[] maxson;
```

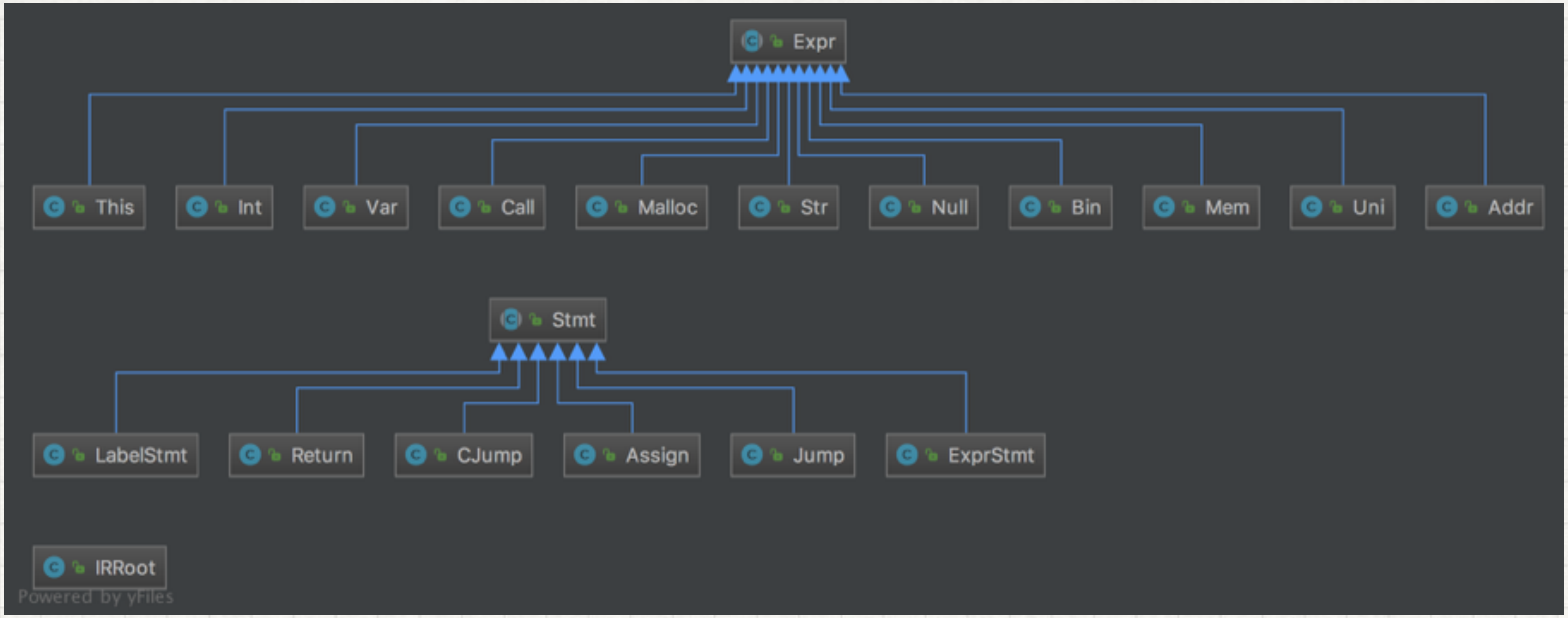
testcase2

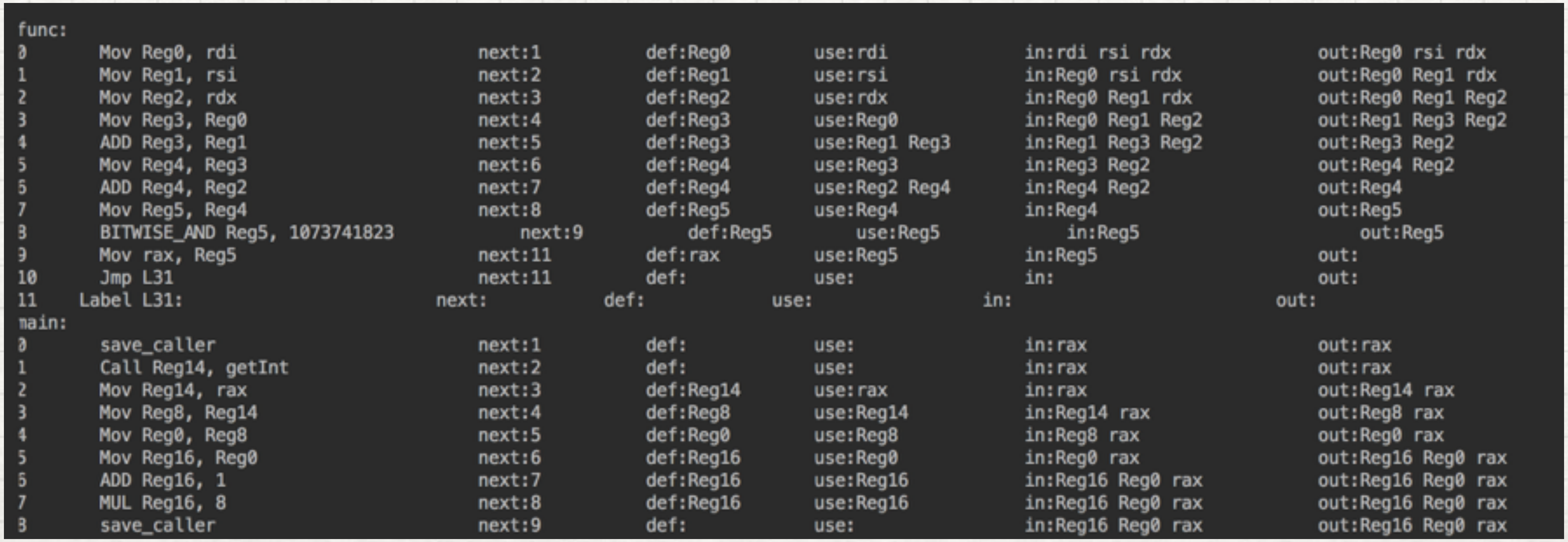
```
void init(){
    e = new Edge[100];
    ne = 0;
    ans = 0;
    size = n;
    visit = new bool[100];
    son = new int[100];
    maxson = new int[100];
    head = new int[100];
    for (i = 0; i <= n; ++ i){
        visit[i] = false;
        son[i] = 0;
        maxson[i] = 0;
        head[i] = 0;
    }
}

int max(int a, int b){
    if (a > b) return a;
    else return b;
}
```

```
void dfs(int u){
    int tmp = 0;
    visit[u] = true;
    int i;
    for (i = head[u]; i != 0; i =
e[i].next){
        int v = e[i].to;
        if (visit[v]==false) {
            dfs(v);
            son[u] = son[u] + (son[v] + 1);
            tmp = max(tmp, son[v] + 1);
        }
    }
    tmp = max(tmp, n - son[u] - 1);
    if (tmp < size || (tmp == size && u <
ans)){
       ans = u;
       size = tmp;
       }
}
```

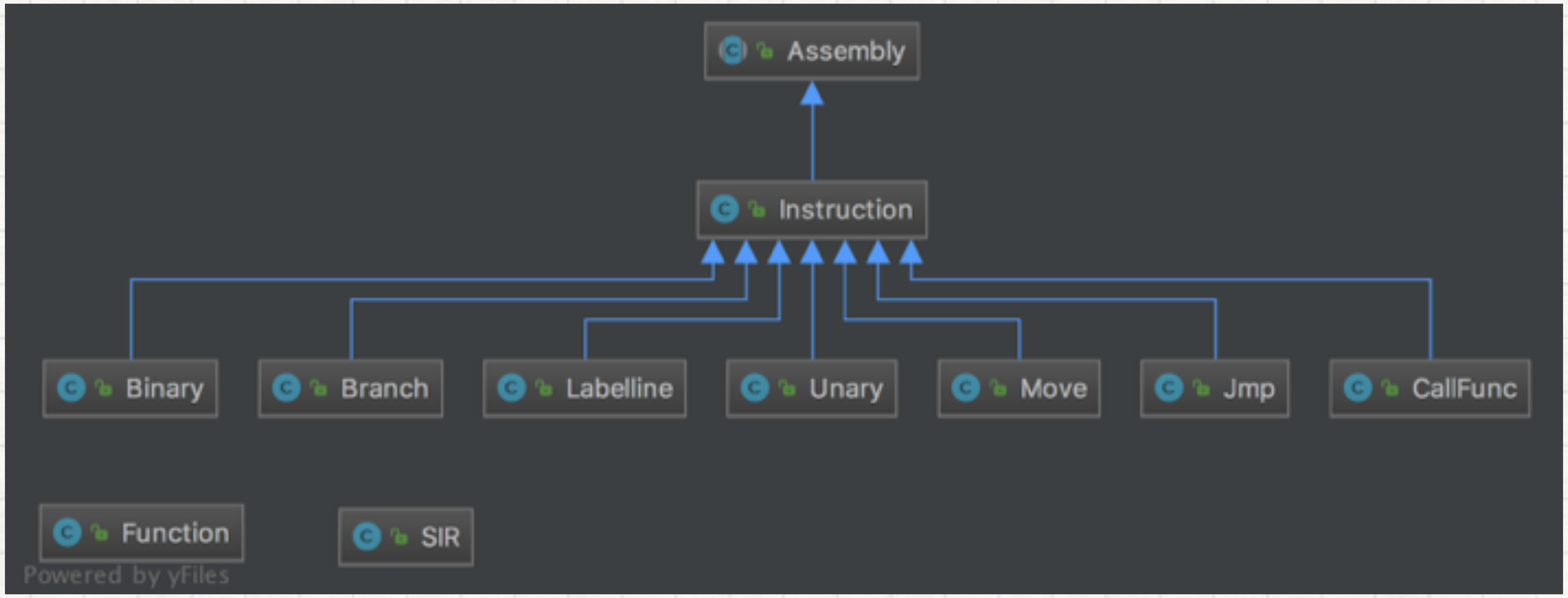

Simpiler Lexer
Simpiler Parser
disassemble a language
AST Builder
AST Printer
Scope Builder
Type Resolver
Dereference Checker
Find out Who I am

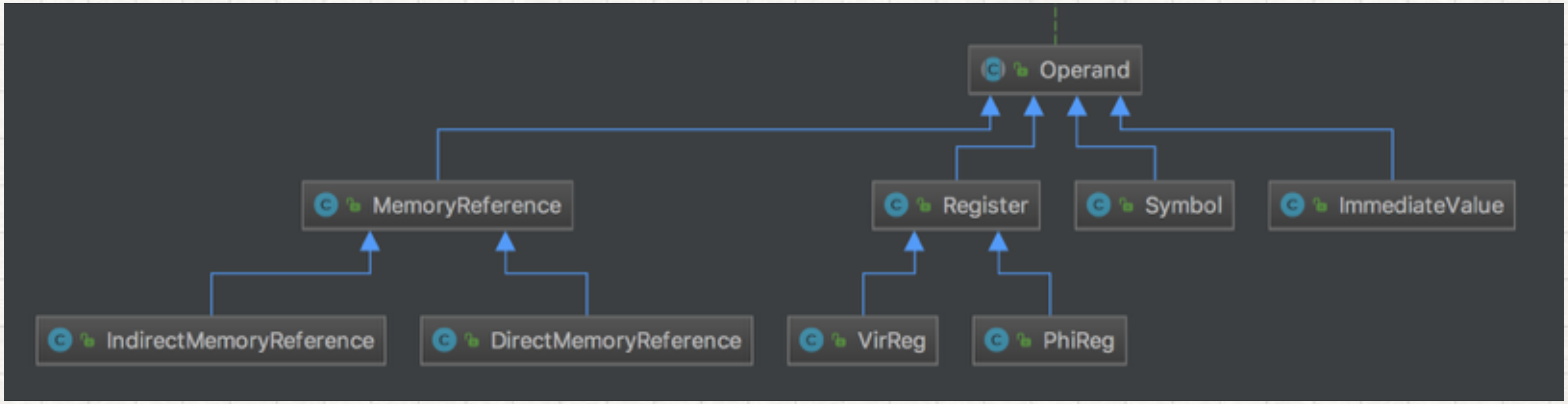

IR Generator
IR Tree
Code Generator
on-the-fly
1-top-of-stack
caching
IR Transformer
Simple Allocator
Code Generator
ASM Printer



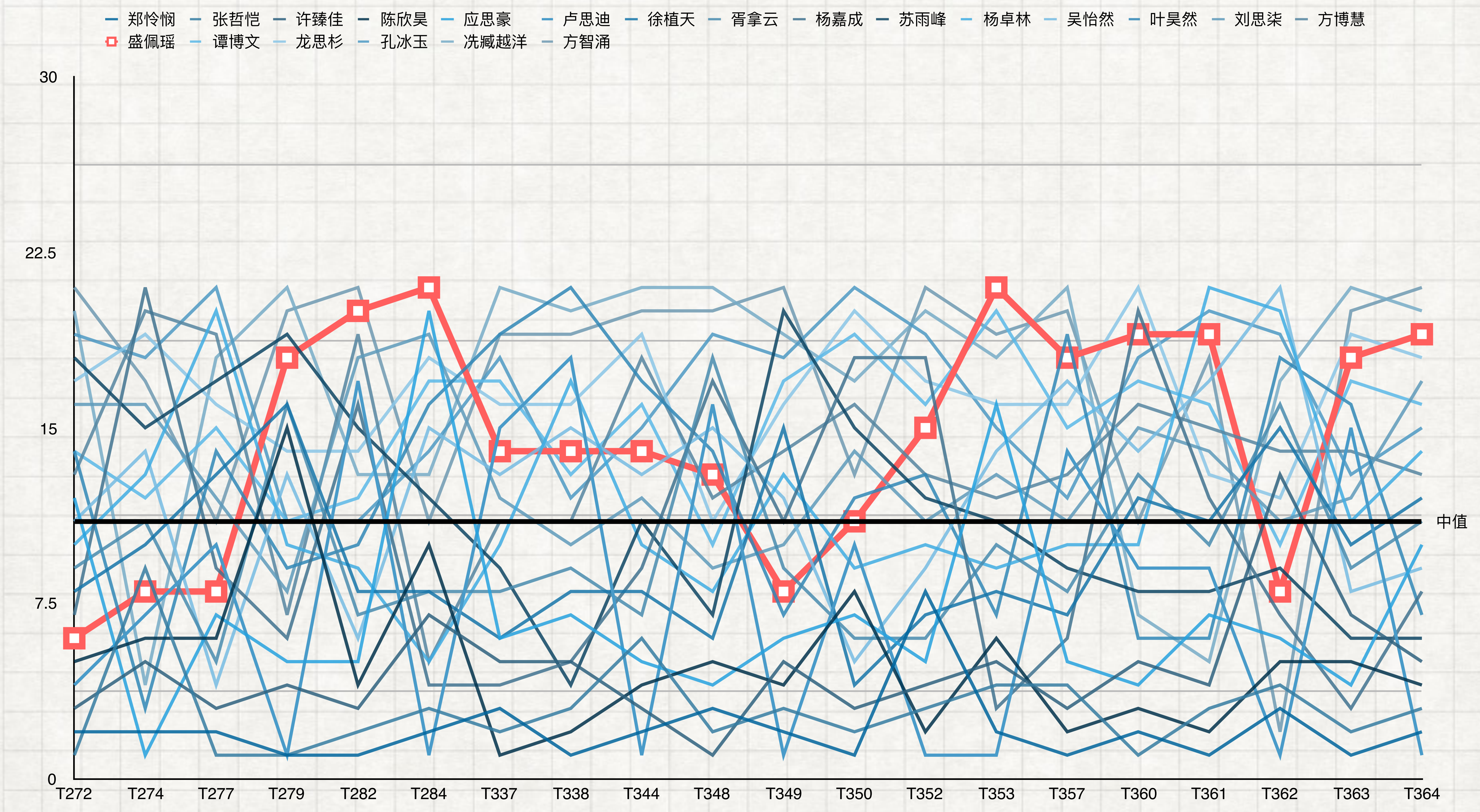


```
func:
0       Mov Reg0, rdi                           next:1          def:Reg0        use:rdi              in:rdi rsi rdx              out:Reg0 rsi rdx
1       Mov Reg1, rsi                           next:2          def:Reg1        use:rsi              in:Reg0 rsi rdx             out:Reg0 Reg1 rdx
2       Mov Reg2, rdx                           next:3          def:Reg2        use:rdx              in:Reg0 Reg1 rdx            out:Reg0 Reg1 Reg2
3       Mov Reg3, Reg0                          next:4          def:Reg3        use:Reg0             in:Reg0 Reg1 Reg2           out:Reg1 Reg3 Reg2
4       ADD Reg3, Reg1                          next:5          def:Reg3        use:Reg1 Reg3        in:Reg1 Reg3 Reg2           out:Reg3 Reg2
5       Mov Reg4, Reg3                          next:6          def:Reg4        use:Reg3             in:Reg3 Reg2                out:Reg4 Reg2
6       ADD Reg4, Reg2                          next:7          def:Reg4        use:Reg2 Reg4        in:Reg4 Reg2                out:Reg4
7       Mov Reg5, Reg4                          next:8          def:Reg5        use:Reg4             in:Reg4                     out:Reg5
8       BITWISE_AND Reg5, 1073741823                next:9          def:Reg5        use:Reg5             in:Reg5                     out:Reg5
9       Mov rax, Reg5                           next:11         def:rax         use:Reg5             in:Reg5                     out:
10      Jmp L31                                 next:11         def:            use:                 in:                         out:
11    Label L31:                            next:           def:            use:                 in:                         out:
main:
0       save_caller                             next:1          def:            use:                 in:rax                      out:rax
1       Call Reg14, getInt                      next:2          def:            use:                 in:rax                      out:rax
2       Mov Reg14, rax                          next:3          def:Reg14       use:rax              in:rax                      out:Reg14 rax
3       Mov Reg8, Reg14                         next:4          def:Reg8        use:Reg14            in:Reg14 rax                out:Reg8 rax
4       Mov Reg0, Reg8                          next:5          def:Reg0        use:Reg8             in:Reg8 rax                 out:Reg0 rax
5       Mov Reg16, Reg0                         next:6          def:Reg16       use:Reg0             in:Reg0 rax                 out:Reg16 Reg0 rax
6       ADD Reg16, 1                            next:7          def:Reg16       use:Reg16            in:Reg16 Reg0 rax           out:Reg16 Reg0 rax
7       MUL Reg16, 8                            next:8          def:Reg16       use:Reg16            in:Reg16 Reg0 rax           out:Reg16 Reg0 rax
8       save_caller                             next:9          def:            use:                 in:Reg16 Reg0 rax           out:Reg16 Reg0 rax
```


Assembly
Instruction
Binary
Branch
Labelline
Unary
Move
Jmp
CallFunc
Function
SIR
Powered by yFiles


Operand
MemoryReference
Register
Symbol
ImmediateValue
IndirectMemoryReference
DirectMemoryReference
VirReg
PhiReg

讲讲自己的编译器在数据集上的表现, 分析原因



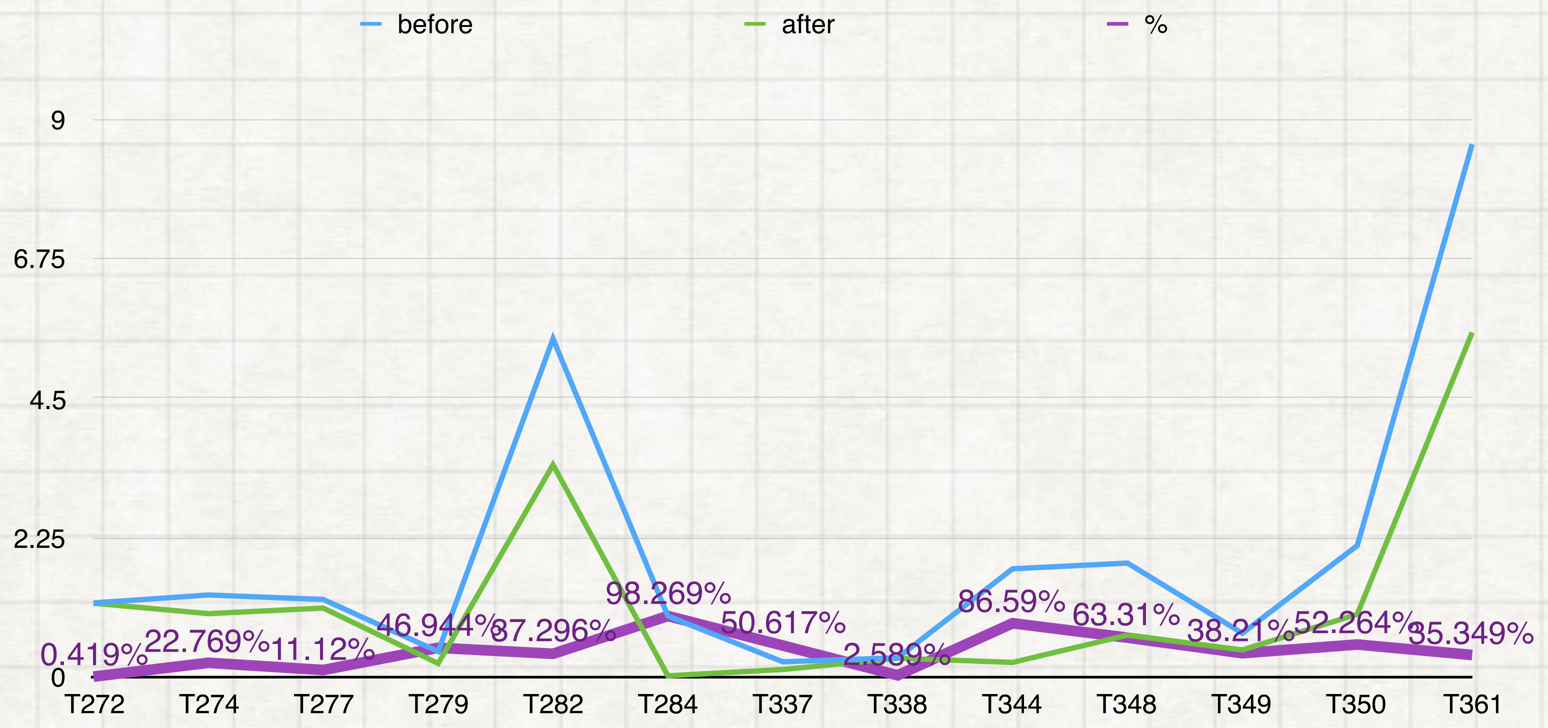
```
T284

for(i = 0; i < k; ++i)
    {
        t = toString(last) + " " + toString(last + 1) + " " + toString(-(last + 2));
        if(i % 100000 == 0)
        {
            println(t);
        }
        last = last + 2;
    }
```

```
T279

void cost_a_lot_of_time(){
    int a = 3100;
    int b = 0;
    int c = 1;
    for (b = 0; b < 100000000; ++b)
        c = c * 2 - c;
    println(toString(a));
}
```

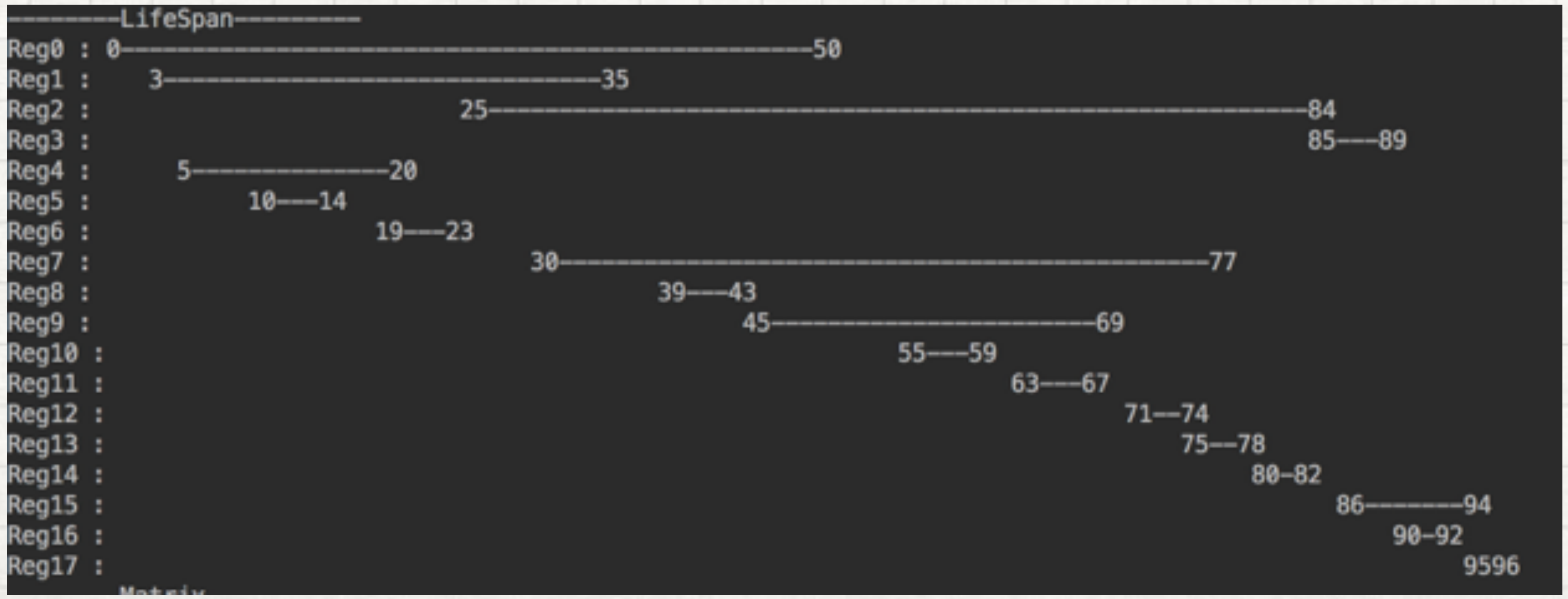
Common Expression Elimination.
Dead Code Elimination
Optimization for loop

```
for(i = 0; i < n; ++i)
    for(j = 0; j < n; ++j){
        for(k = 0; k < n; ++k){
            if(j >= i){
                g[i][j] = func(g[i][j], f[i][k], f[k][j]);
                g_useless[i][j] = func(g[i][j], f[i][k], f[k][j]);
                g_useless[i][j] = func(g[i][j], f[i][k], f[k][j]);
                g_useless[i][j] = func(g[i][j], f[i][k], f[k][j]);
            }
        }
    }
```

- `print(toString()) => printInt()`
- `Dead for, if-for`
- `Simple Inline`



- `split reg to life span`



```
--------LifeSpan---------
Reg0 : 0----------------------------------------------------50
Reg1 :    3---------------------------------35
Reg2 :                            25--------------------------------------------------------------84
Reg3 :                                                                                            85---89
Reg4 :      5---------------20
Reg5 :           10---14
Reg6 :                     19---23
Reg7 :                                30--------------------------------------------------77
Reg8 :                                          39---43
Reg9 :                                                45-------------------------69
Reg10 :                                                           55---59
Reg11 :                                                                   63---67
Reg12 :                                                                           71--74
Reg13 :                                                                               75--78
Reg14 :                                                                                     80-82
Reg15 :                                                                                            86--------94
Reg16 :                                                                                                90-92
Reg17 :                                                                                                      9596
        Matrix
```