CMPS104A: Code Generation Tutorial

Christopher Schuster

University of California, Santa Cruz

December 3, 2013

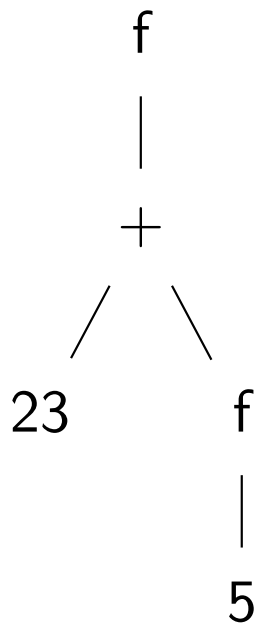
Outline

Code generation example (Assembler)

```
f(23 + f(5));
```

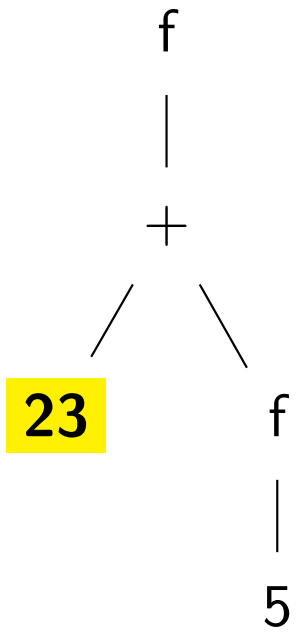
Code generation example (Assembler)

```
f(23 + f(5));
```



Code generation example (Assembler)

```
f(23 + f(5));
```

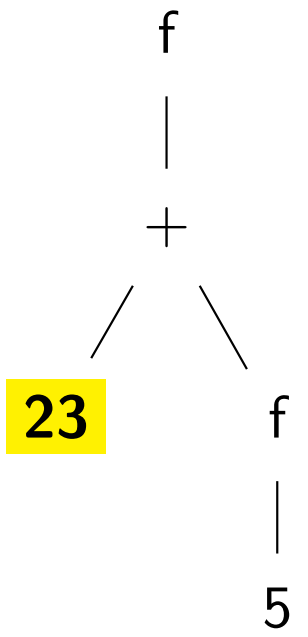



```
MOV EAX 23
```

```
EAX: 23
EBX:
Stack:
```

Code generation example (Assembler)

```
f(23 + f(5));
```

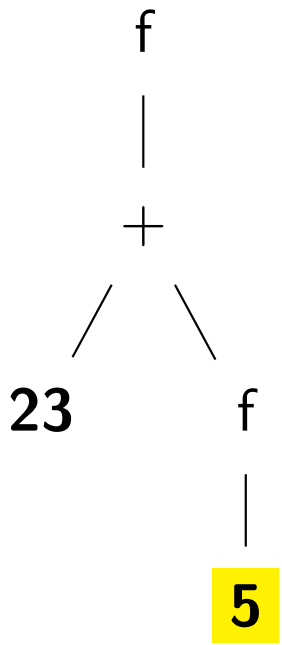


EAX: 23
EBX:
Stack:
[23]

```
MOV EAX 23
PUSH EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

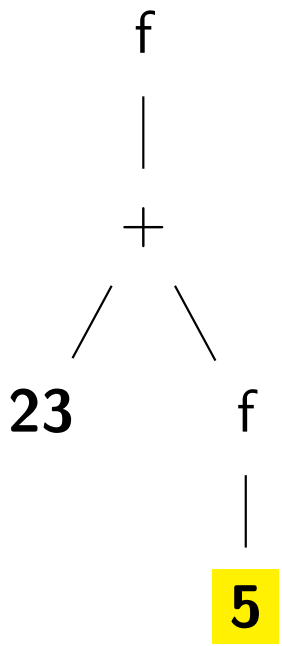


EAX: 5
EBX:
Stack:
[23]

```
MOV EAX 23
PUSH EAX
MOV EAX 5
```

Code generation example (Assembler)

```
f(23 + f(5));
```

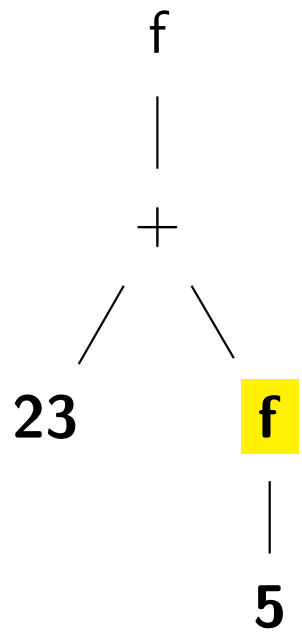



```
EAX:  5
EBX:
Stack:
[23,5]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

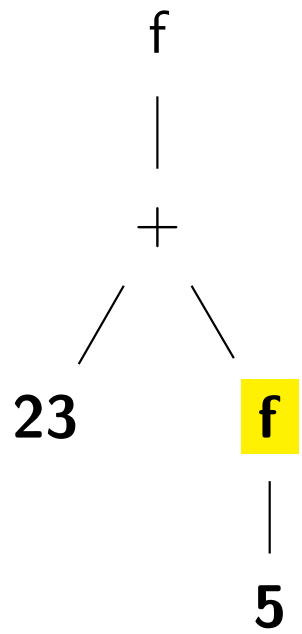



```
EAX:  5
EBX:
Stack:
[23]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

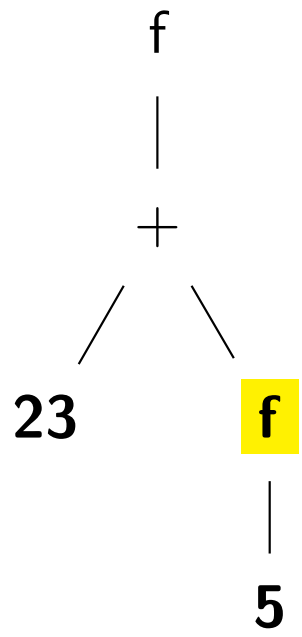



```
EAX:    6
EBX:
Stack:
[23]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
```

Code generation example (Assembler)

```
f(23 + f(5));
```

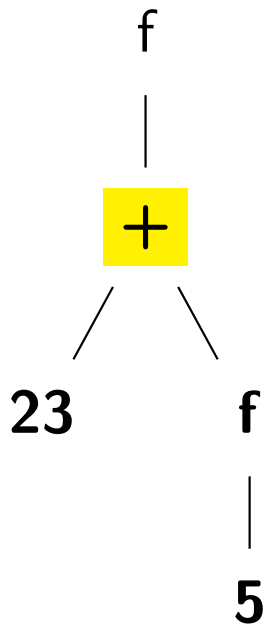



```
EAX:    6
EBX:
Stack:
[23,6]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

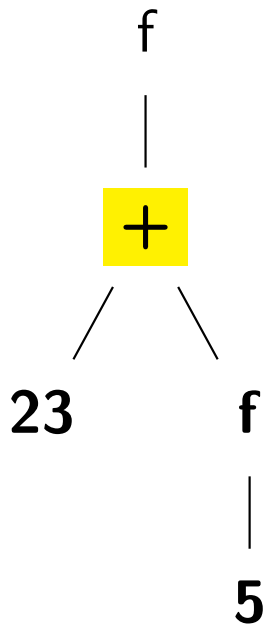



```
EAX:    6
EBX:
Stack:
[23]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

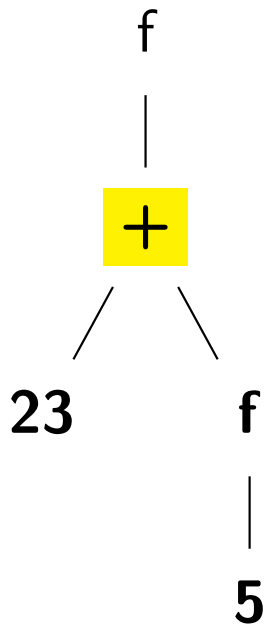



```
EAX:   6
EBX: 6
Stack:
[23]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

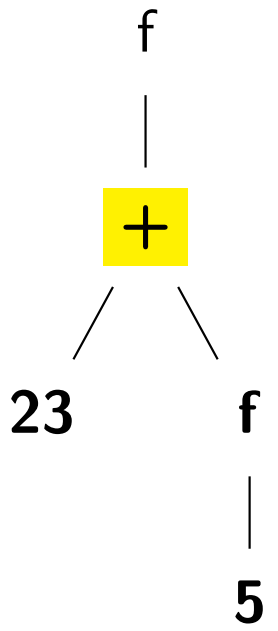



```
EAX:    23
EBX: 6
Stack:
[]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

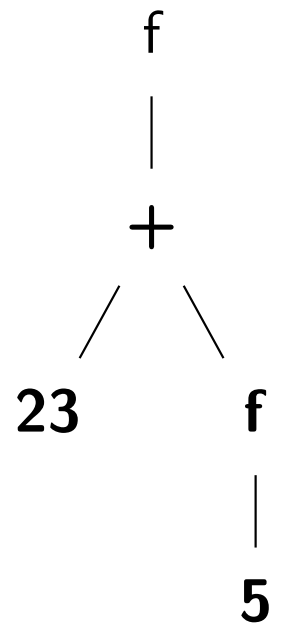



```
EAX:      29
EBX: 6
Stack:
[]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

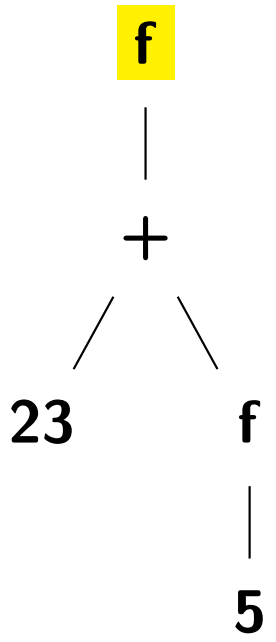



```
EAX:      29
EBX: 6
Stack:
[29]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

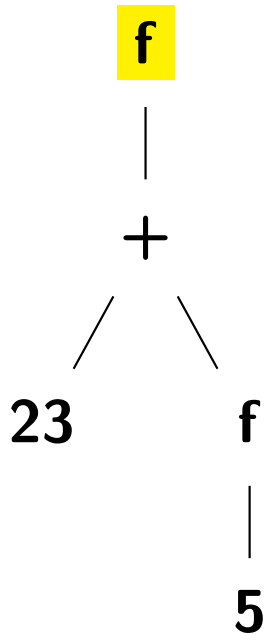



```
EAX:      29
EBX: 6
Stack:
[]
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
```

Code generation example (Assembler)

```
f(23 + f(5));
```

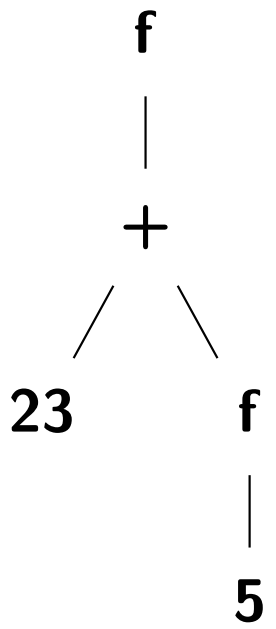


EAX: 30
EBX: 6
Stack:
[]

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

Code generation example (Assembler)

```
f(23 + f(5));
```

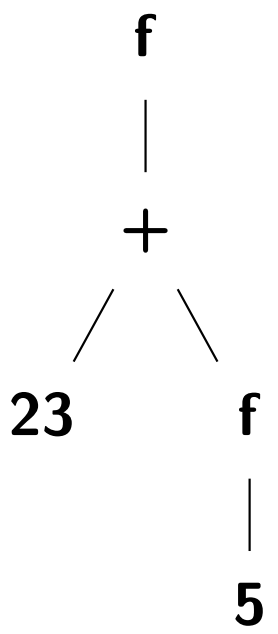



```
EAX:
EBX: 6
Stack:
```

```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

Code generation example (Assembler → oil)

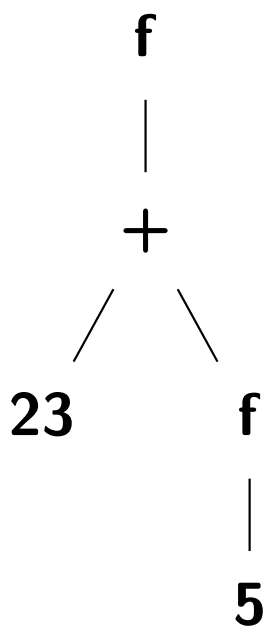
```
f(23 + f(5));
```




```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

Code generation example (Assembler → oil)

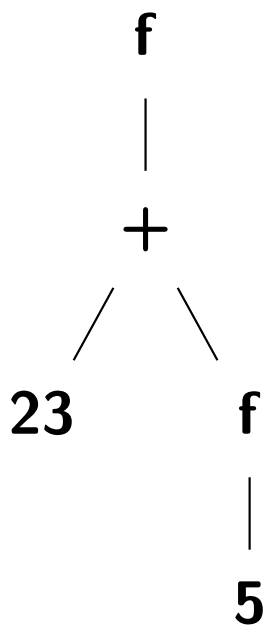
```
f(23 + f(5));
```




```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

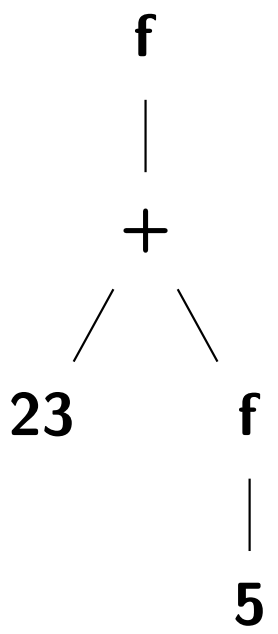



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

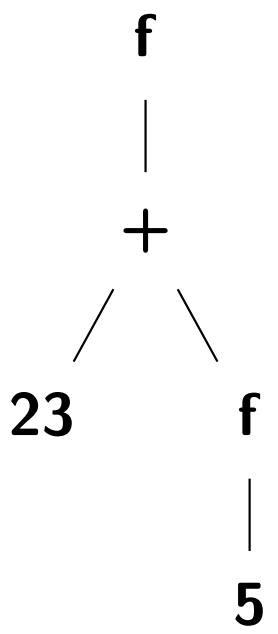



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

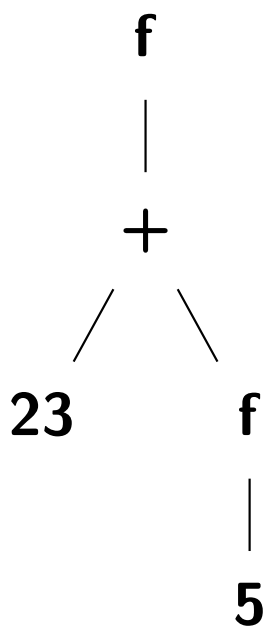



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

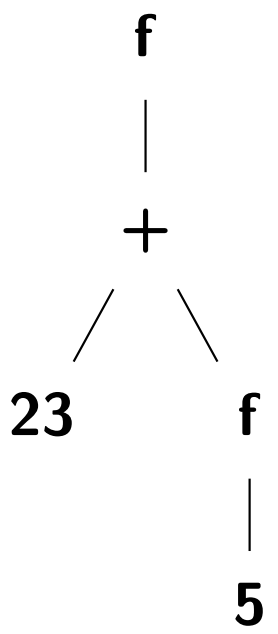



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

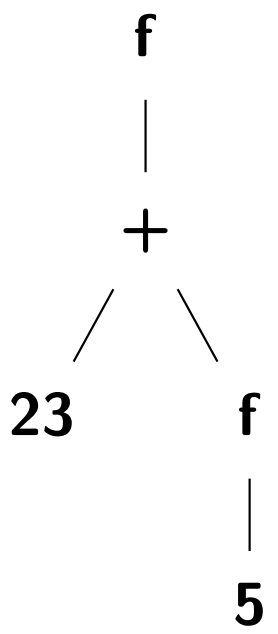



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

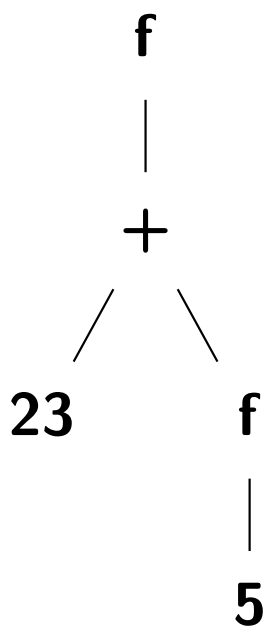



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

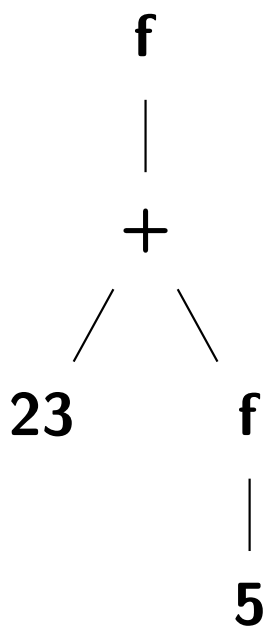



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

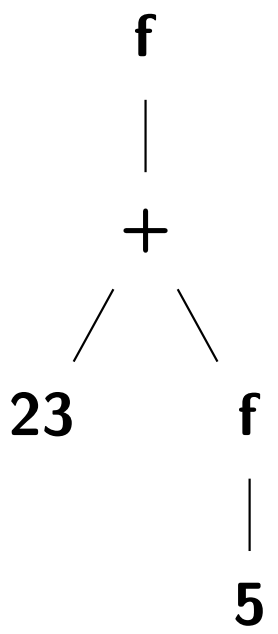



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

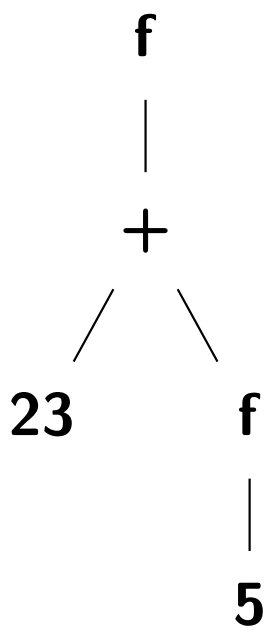



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

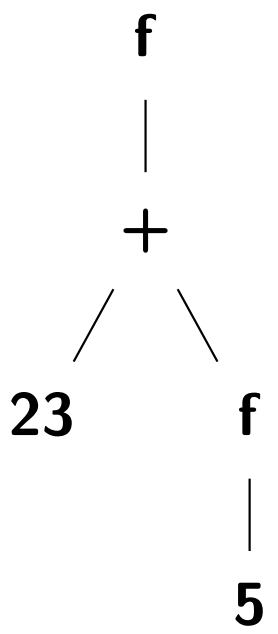



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

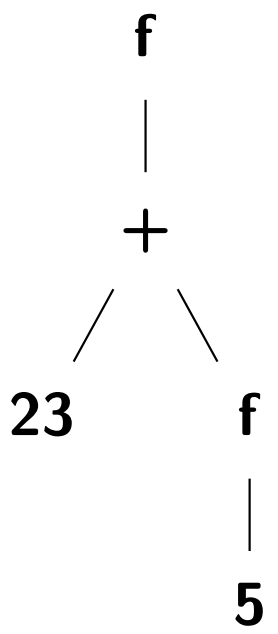



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

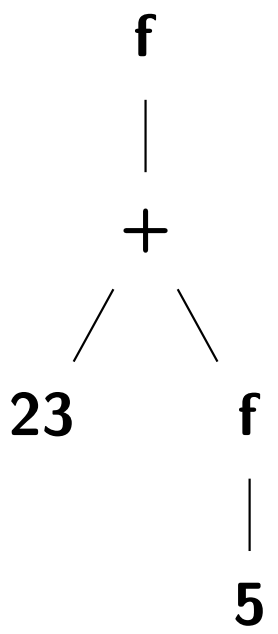



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

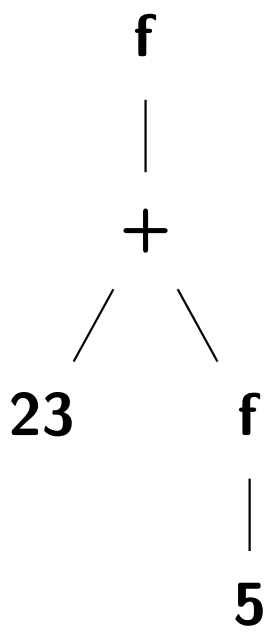



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

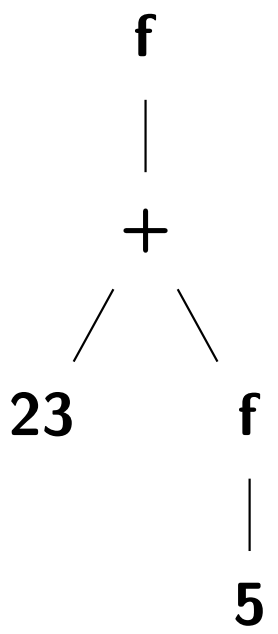



```
MOV EAX 23
PUSH EAX
MOV EAX 5
PUSH EAX
POP EAX
CALL f
PUSH EAX
POP EAX
MOV EBX EAX
POP EAX
ADD EAX EAX EBX
PUSH EAX
POP EAX
CALL f
```

```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

Code generation example (Assembler → oil)

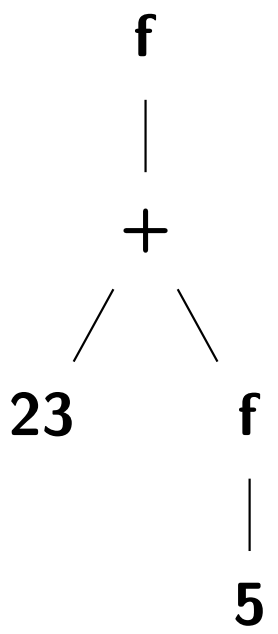
```
f(23 + f(5));
```




```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

Code generation example (Assembler → oil)

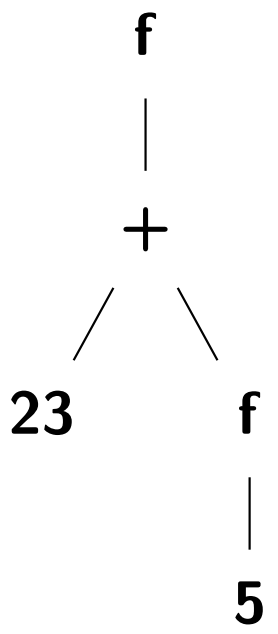
```
f(23 + f(5));
```




```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

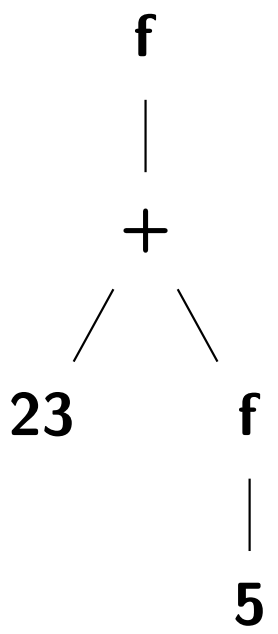



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

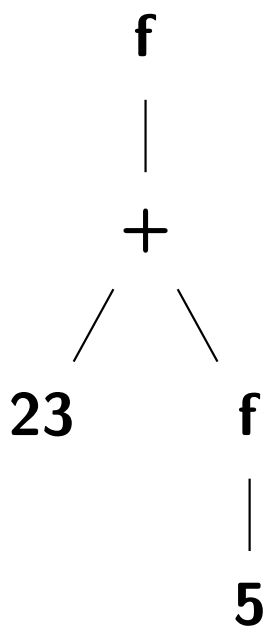



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

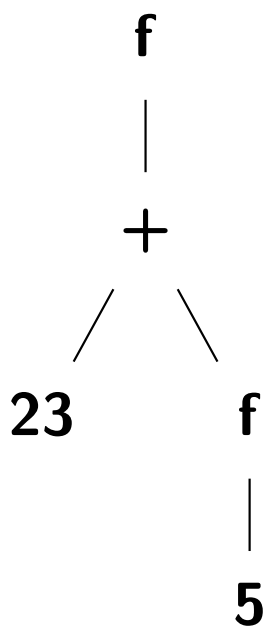



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

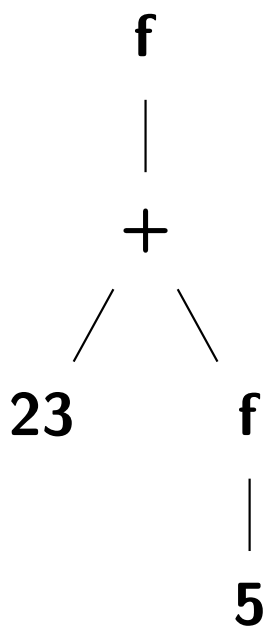



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

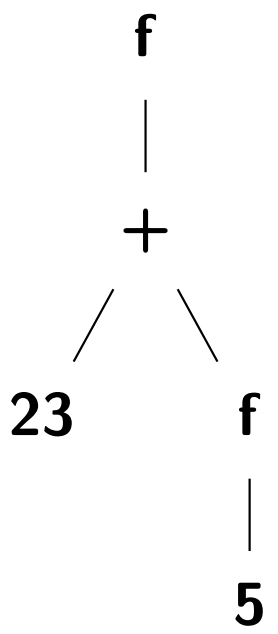



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

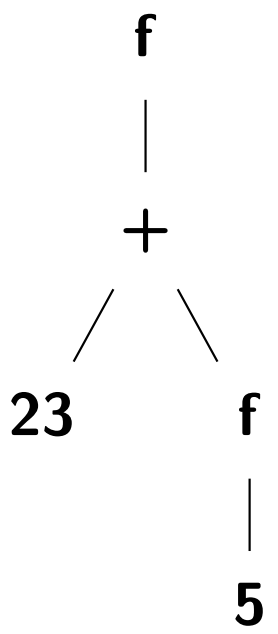



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

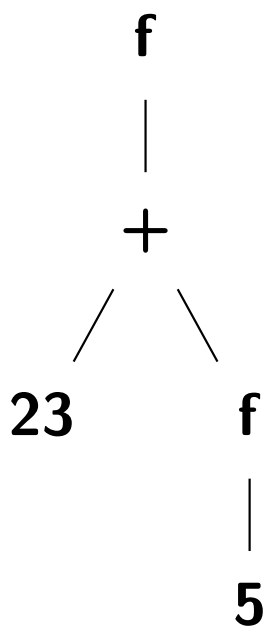



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

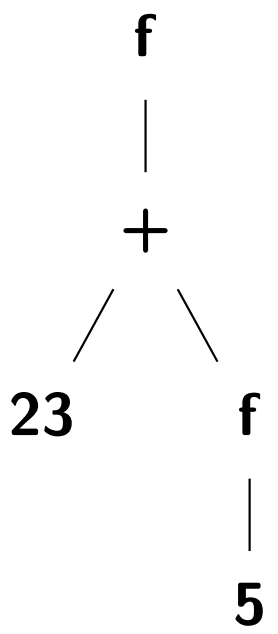



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

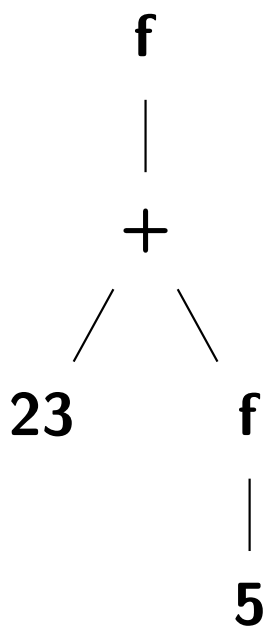



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

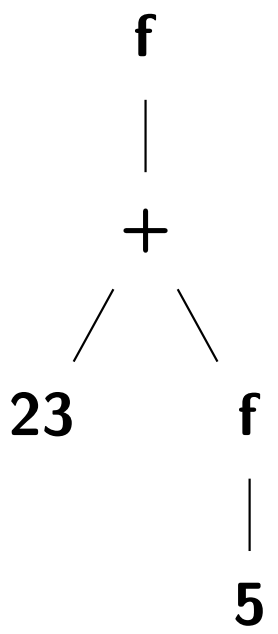



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

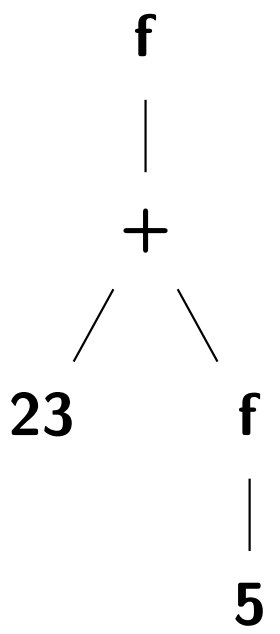



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
t4 = t1 + t3
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

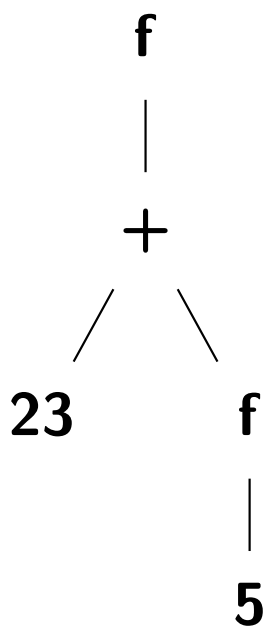



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
t4 = t1 + t3
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```

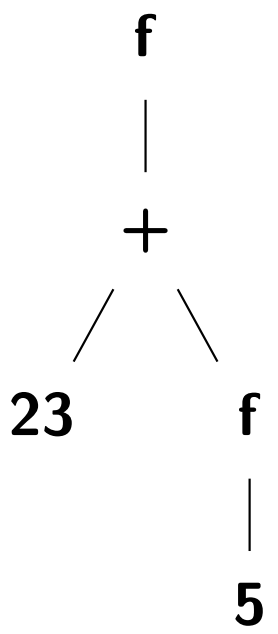



```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

```
t1 = 23
t2 = 5
t3 = f(t2)
t4 = t1 + t3
```

Code generation example (Assembler → oil)

```
f(23 + f(5));
```




```
EAX = 23
t1 = EAX
EAX = 5
t2 = EAX
EAX = t2
EAX = f(EAX)
t3 = EAX
EAX = t2
EBX = EAX
EAX = t2
EAX = EAX + EBX
t4 = EAX
EAX = t4
EAX = f(EAX)
```

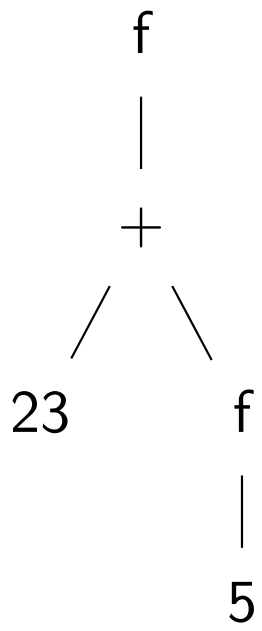
```
t1 = 23
t2 = 5
t3 = f(t2)
t4 = t1 + t3
t5 = f(t4)
```

Code generation example (oil)

```
f(23 + f(5));
```

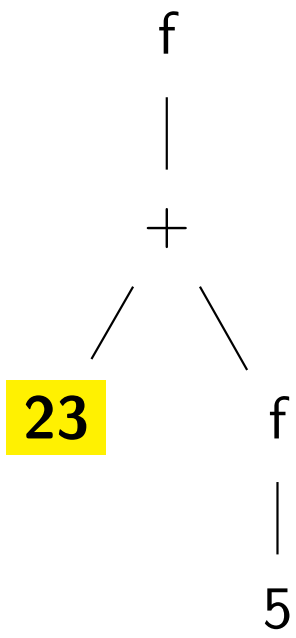
Code generation example (oil)

```
f(23 + f(5));
```



Code generation example (oil)

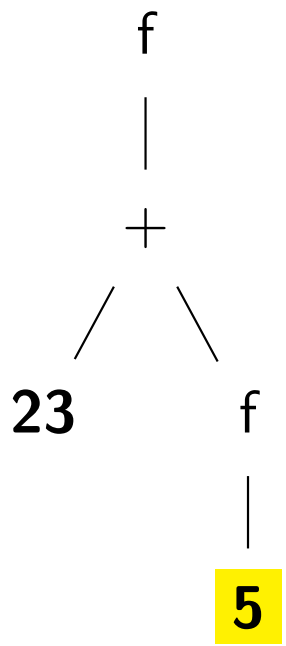
```
f(23 + f(5));
```




```
int i1 = 23;
```

Code generation example (oil)

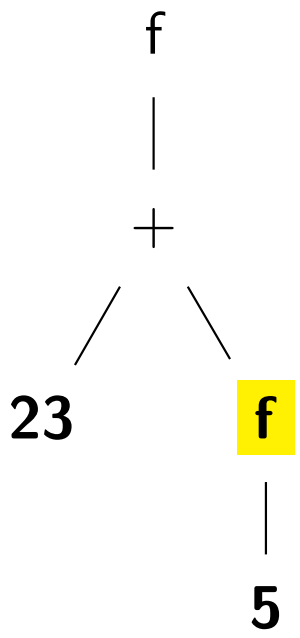
```
f(23 + f(5));
```




```
int i1 = 23;
int i2 = 5;
```

Code generation example (oil)

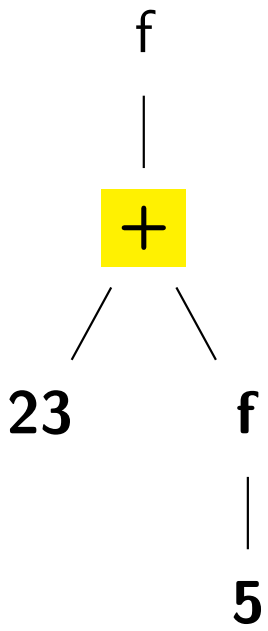
```
f(23 + f(5));
```




```
int i1 = 23;
int i2 = 5;
int i3 = f(i2);
```

Code generation example (oil)

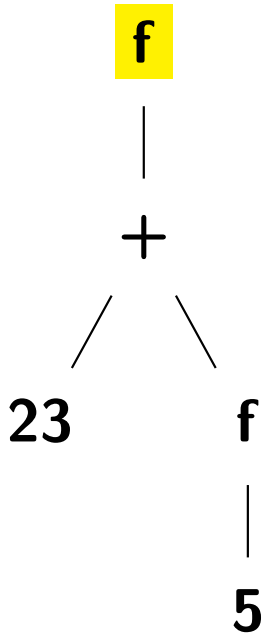
```
f(23 + f(5));
```




```
int i1 = 23;
int i2 = 5;
int i3 = f(i2);
int i4 = i1 + i3;
```

Code generation example (oil)

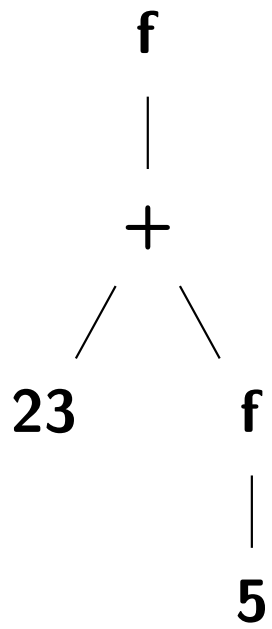
```
f(23 + f(5));
```




```
int i1 = 23;
int i2 = 5;
int i3 = f(i2);
int i4 = i1 + i3;
int i5 = f(i4);
```

Code generation example (oil)

```
f(23 + f(5));
```




```
int i1 = 23;
int i2 = 5;
int i3 = f(i2);
int i4 = i1 + i3;
int i5 = f(i4);
```

Code generation implementation (1/3)

```
class Node {
  virtual string codegen() = 0;
};
```

Code generation implementation (1/3)

```
class Node {
  virtual string codegen() = 0;
};
class Block : public Node {
  vector<Node*> nodes;
public:
  string codegen() {
    for (Node* n in this->nodes) {
      cout << n->codegen() << ";" << endl;
    }
    return "";
  }
};
```

Code generation implementation (2/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
};
```

Code generation implementation (2/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen() {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    return left + " + " + right;
  }
};
```

Code generation implementation (2/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen() {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    return left + " + " + right;
  }
};
class Const : public Node {
  string codegen() {
    return this->val;
  }
}
```

Code generation implementation (2/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen() {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    return left + " + " + right;
  }
};
class Const : public Node {
  string codegen() {
    return this->val;
  }
}

new BinOp(new BinOp(new Const(23),new Const(5)),new Const(10))
```

Code generation implementation (2/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen() {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    return left + " + " + right;
  }
};
class Const : public Node {
  string codegen() {
    return this->val;
  }
}

new BinOp(new BinOp(new Const(23),new Const(5)),new Const(10))
-> 23 + 5 + 10
```

Code generation implementation (3/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
```

Code generation implementation (3/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
  static int counter;
```

Code generation implementation (3/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
  static int counter;
  string save(string code) {
    char buffer[10];
    sprintf(buffer, "t%d", ++Node::counter);
    cout << "int " << buffer << " = " << code << ";" << endl;
    return buffer;
  }
};
```

Code generation implementation (3/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
  static int counter;
  string save(string code) {
    char buffer[10];
    sprintf(buffer, "t%d", ++Node::counter);
    cout << "int " << buffer << " = " << code << ";" << endl;
    return buffer;
  }
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen(bool save) {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    string code = left + " + " + right;
    return save ? this->save(code) : code;
  }
};
```

Code generation implementation (3/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
  static int counter;
  string save(string code) {
    char buffer[10];
    sprintf(buffer, "t%d", ++Node::counter);
    cout << "int " << buffer << " = " << code << ";" << endl;
    return buffer;
  }
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen(bool save) {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    string code = left + " + " + right;
    return save ? this->save(code) : code;
  }
};
new BinOp(new BinOp(new Const(23),new Const(5)),new Const(10)))
```

Code generation implementation (3/3)

```
class Node {
public:
  virtual string codegen(bool temp) = 0;
  static int counter;
  string save(string code) {
    char buffer[10];
    sprintf(buffer, "t%d", ++Node::counter);
    cout << "int " << buffer << " = " << code << ";" << endl;
    return buffer;
  }
};
class BinOp : public Node {
  Node* left;
  Node* right;
public:
  string codegen(bool save) {
    string left = this->left->codegen(true);
    string right = this->right->codegen(true);
    string code = left + " + " + right;
    return save ? this->save(code) : code;
  }
};
new BinOp(new BinOp(new Const(23),new Const(5)),new Const(10)))
-> t1 = 23;
   t2 = 5;
   t3 = t1 + t2;
   t4 = 10;
   t5 = t3 + t4;
```

Code generation implementation (using `switch`)

```
struct astree { ... };
int counter = 0;
```

Code generation implementation (using switch)

```
struct astree { ... };
int counter = 0;
string save_in_reg(string code) {
  char buffer[10];
  sprintf(buffer, "t%d", ++counter);
  printf("int %s = %s;\n", buffer, code.c_str());
  return buffer;
}
```

Code generation implementation (using switch)

```
struct astree { ... };
int counter = 0;
string save_in_reg(string code) {
  char buffer[10];
  sprintf(buffer, "t%d", ++counter);
  printf("int %s = %s;\n", buffer, code.c_str());
  return buffer;
}
string codegen(Node* node, bool save) {
  string code = "";
  switch(node->getSymbol()) {
    case TOK_ROOT:
      for (Node* child in node->children) {
        printf("%s;\n", codegen(child).c_str());
      }
      break;
    case TOK_BINOP:
      string left = codegen(node->children[0], true);
      string right = codegen(node->children[2], true);
      code = left + " + " + right;
      break;
    case TOK_INTCON:
      code = node->lexinfo;
    ...
  }
  return save ? save_in_reg(code) : code;
};
```