

# Midterm Solution

ECEN 4553 & 5013, CSCI 4555 & 5525, Fall 2007

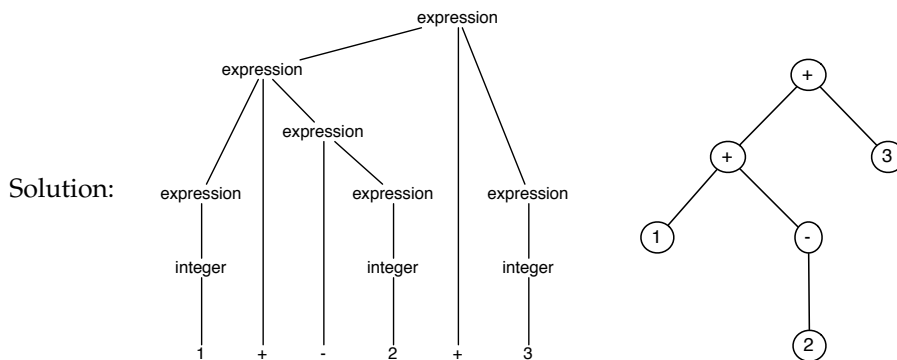
- (10 points) Draw a parse tree and an abstract syntax tree (AST) for the following program, given the following grammar and using the precedence rules of Python. The AST should match what the Python compiler module would produce.

Grammar:

```
expression ::= integer | "-" expression | expression "+" expression
```

Program:

```
1 + - 2 + 3
```



- (15 points) Write down the PLY parser specification for the above grammar, including the construction of the AST nodes but not the lexer specification. Assume that the tokens INT, PLUS, and MINUS have been defined. Write the `p_` functions and the definition for the precedence variable. The actions inside the `p_` should build the appropriate Python AST nodes.

Solution:

```
precedence = (
    ('left', '+'),
    ('right', 'UMINUS')
)

def p_plus(t):
    'expression : expression PLUS expression'
    t[0] = Add((t[1], t[3]))

def p_uminus(t):
    'expression : MINUS expression %prec UMINUS'
    t[0] = UnarySub(t[2])

def p_int(t):
    'expression : INT'
    t[0] = Const(t[1])
```

3. (15 points) Write the C code that your compiler generates for the following  $P_1$  program, including the functions or declarations needed from the runtime support, except for the following function that prints any `pyobj`, which you can assume is already implemented.

```
void print_pyobj(pyobj);
```

Assume that your compiler is not doing any type analysis or specialization. Remember that  $P_1$  deals with three kinds of values: normal integers, floating-point numbers, and Booleans.

```
print - 2
```

Solution:

```
enum type { INT, FLOAT, BOOL };

typedef struct pyobj_struct {
    enum type tag;
    union {
        int i;
        double f;
        short b;
    } u;
} pyobj;

pyobj make_int(int x) {
    pyobj v;
    v.tag = INT;
    v.u.i = x;
    return v;
}

pyobj negate(pyobj x) {
    switch(x.tag) {
        case BOOL:
            x.u.b = - x.u.b
            break;
        case INT:
            x.u.i = - x.u.i
            break;
        case FLOAT:
            x.u.f = - x.u.f
            break;
        default:
            exit(-1);
    }
    return x;
}

int main() {
    print_pyobj(negate(make_int(2)));
    return 0;
}
```

4. (10 points) Convert the following program to Static Single-Assignment (SSA) form, inserting  $\phi$  nodes where necessary.

```
x = 4
def foo(y):
    return x + y
z = 21
if True:
    def foo(y):
        return z + y
else:
    z = 3
print foo(z)
```

Solution:

```
x0 = 4
def foo0(y0):
    return x0 + y0
z0 = 21
if True:
    def foo1(y1):
        return z0 + y1
else:
    z1 = 3
foo2 =  $\phi$ (foo1, foo0)
z2 =  $\phi$ (z0, z1)
print foo2(z2)
```

5. (10 points) Perform type analysis and write down the type that should be assigned to each variable.

```
x0 = 0
y0 = x0 + 2
while True:
    y1 =  $\phi$ (y0, y2)
    if x0 < y1:
        y2 = x0 - 0.5
    else:
        break
```

Solution:

```
x0 : int
y0 : int
y1 : pyobj
y2 : float
```

6. (10 points) What is the output of the following  $P_3$  program?

```
y = [3,5]
x = [1,1,y] + y
y = [8,2]
x[2] = 2
z = x + [8,13,21]
x[0] = 0
print z
```

Solution:

```
[1,1,2,3,5,8,13,21]
```

7. (10 points) Fill in the empty spaces below in the definition of a function that computes the free variables of an expression.

```
def free_vars_in_expr(n):
    if isinstance(n, Const):
        return Set([])
    elif isinstance(n, Name):
        if n.name == 'True' or n.name == 'False':
            return Set([])
        else:
            return Set([n.name])
    elif isinstance(n, Add):
        return free_vars_in_expr(n.left) | free_vars_in_expr(n.right)
    elif isinstance(n, UnaryAdd):
        return free_vars_in_expr(n.expr)
    elif isinstance(n, CallFunc):
        return free_vars_in_expr(n.node) \
            | reduce(lambda a,b: a | b, [free_vars_in_expr(e) for e in n.args], Set([]))
    elif isinstance(n, Lambda):
        return free_vars_of_expr(n.code) - Set(n.argnames)
    else:
        raise Exception('Error in free_vars_in_expr: unrecognized AST node')
```

8. (5 points) What is the output of the following program?

```
T = lambda thn: lambda els: thn
print T(42)(24)
```

Solution:

```
42
```

9. (15 points) Closure convert the above Python program to an equivalent Python program where no function has free variables. Do not perform any optimizations such as constant folding.

```
def anon_0(fvs, els):
    thn = fvs[0]
    return thn

def anon_1(fvs, thn):
    return [anon_0, [thn]]

T = [anon_1, []]

tmp = T[0](T[1], 42)
print tmp[0](tmp[1], 24)
```