Recursive-Descent Parser

Authors: Sara Ackerman, Koren Spell, Garrett Williams
Reviewers: Mallory Anderson, Jermy Appiah, Luke Hawranick

```
**accept**(terminal) {
    if given terminal is next in input:
        advance in input
        return true
    else if its not:
        return false

**expect**(terminal) {
    if terminal matches:
        advance
    else on a mismatch:
        reject

**parse**(input) {
    advance
    //call start production
    stmt()

    //accepts token stream if token stream is empty after all productions
    if input == NULL/EMPTY):
        accept
    else:
        reject
}

**stmt**() {
    if accept(int_keyw):
        expect(identifier)
        expect(=)
        expr()
        expect(;)
        stmt()
else if accept(float_keyw):
        expect(identifier)
        expect(=)
        expr()
        expect(;)
        stmt()
else if accept(identifier):
        factors()
        terms()
        compares()
        equals()
        expect(;)
        stmt()
else if accept(int):
        factors()
        terms()
        compares()
        equals()
        expect(;)
        stmt()
else if accept(float):
        factors()
        terms()
        compares()
        equals()
        expect(;)
        stmt()
else if accept(open_paren):
        expr()
        expect(close_paren)
        factors()
        terms()
        compares()
```

```
                equals()
                expect(;)
                stmt()
        else if accept(if_keyw):
                expect(open_paren)
                expr()
                expect(close_paren)
                block()
                else()
                stmt()
        else if accept(for_keyw):
                expect(open_paren)
                pre()
                expr()
                expect(;)
                expr()
                expect(close_paren)
                block()
                stmt()
        else if accept(while_keyw):
                expect(open_paren)
                expr()
                expect(close_paren)
                block()
                stmt()
        else if accept(close_brack) || accept(end_of_input):
                pass
        else:
                reject
}

block() {
        expect({)
        stmt()
        expect(})
```

```
}

else() {
    expect(else_keyw)
    block()
    if accept(int_keyw) || accept(float_keyw) || accept(identifier) || accept(int)
    || accept(float) || accept(open_paren) || accept(if_keyw) || accept(for_keyw)
    || accept(while_keyw):
        pass
    else:
        reject
}

pre() {
    if not accept(int_keyw):
        accept(float_keyw)
    expect(identifier)
    expect(=)
    expr()
    expect(;)
}

inc_op() {
    if accept(++):
        pass
    else if accept(--):
        pass
    else:
        reject
}

inc() {
    if accept(identifier):
        inc_op()
    else:
        reject
}

expr() {
    if accept(identifier):
        factors()
        terms()
        compares()
        equals()
    else if accept(int_keyw):
        factors()
        terms()
        compares()
        equals()
    else if accept(float_keyw):
        factors()
        terms()
        compares()
        equals()
    else if accept(open_paren):
        expr()
        expect(close_paren)
        factors()
        terms()
        compares()
        equals()
    else:
        reject
}

assigns(){
    If accept(=):
        assign()
        assigns()
    else:
        reject
}

assign(){
    if accept(identifier):
        factors()
        terms()
        compares()
        equals()
    else if accept(int):
        factors()
        terms()
        compares()
        equals()
    else if accept(float):
        factors()
        terms()
        compares()
        equals()
    else if accept(open_paren):
        expr()
        expect(close_paren)
        factors()
        terms()
        compares()
        equals()
}

equals(){
    if accept(==):
        equal()
        equals()
    else if accept(!=):
        equal()
        equals()
    else:
        reject

equal(){
    if accept(identifier):
        factors()
        terms()
        compares()
    else if accept(int):
        factors()
        terms()
        compares()
    else if accept(float):
        factors()
        terms()
        compares()
    else if accept(open_paren):
        expr()
        expect(close_paren)
        factors()
        terms()
        compares()
}

compares(){
    if accept(<):
        compare()
        compares()
    else if accept(>):
        compare()
        compares()
    else if accept(<=):
        compare()
        compares()
    else if accept(>=):
        compare()
        compares()
    else:
        reject
}
compare(){
    if accept(identifier):
        factors()
        terms()
    else if accept(int):
        factors()
        terms()
    else if accept(float):
        factors()
        terms()
    else if accept(open_paren):
        expr()
        expect(close_paren)
        factors()
        terms()
}
terms(){
    if accept(+):
        term()
        terms()
    else if accept(-):
        term()
        terms()
    else:
        reject

term(){
    if accept(identifier):
        factors()
    else if accept(int):
        factors()
    else if accept(float):
        factors()
    else if accept(open_paren):
        expr()
        expect(close_paren)
        factors()
}
factors(){
    if accept(*):
        factor()
        factors()
    else if accept(/):
        factor()
        factors()
    else:
        reject
}

factor(){
    if accept(identifier):
        pass
    else if accept(int):
        pass
    else if accept(float):
        pass
    else if accept(open_paren):
        expr()
        expect(close_paren)
    else if accept(-):
        value()
}

value() {
    if (accept(int)):
```

```
                pass
        else if accept(float)):
                pass
        else if accept(open_paren):
                expr()
                expect(close_paren)
}
```