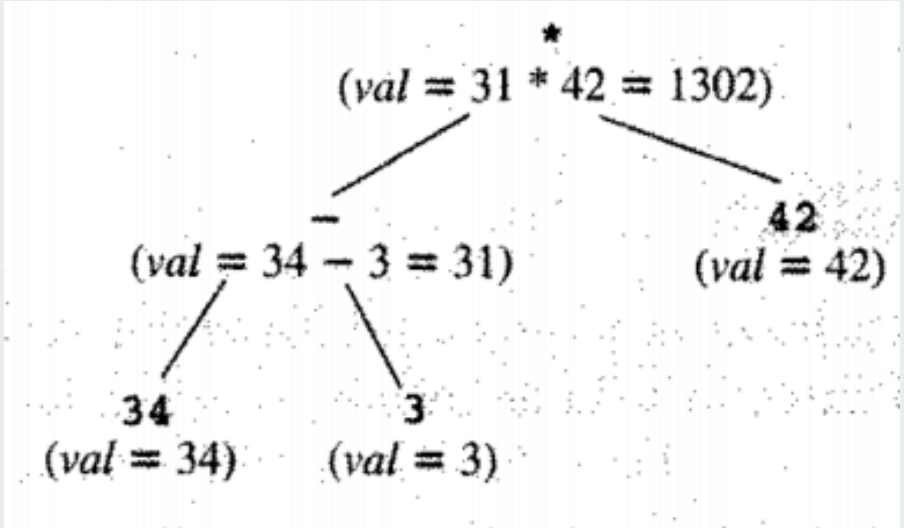
# A Desk Calculator




BJARNE STROUSTRUP

# Token Stream

```
class Token_stream {
public:
    Token_stream(istream& s) : ip{&s}, owns{false} { }
    Token_stream(istream* p) : ip{p}, owns{true} { }
    ~Token_stream() { close(); }
    Token get();
    Token& current(){
        return ct;
    }
    void set_input(istream& s) { close(); ip = &s; owns=false; }
    void set_input(istream* p) { close(); ip = p; owns = true; }
private:
    void close() { if (owns) delete ip; }
    istream* ip;
    bool owns;
    Token ct {Kind::end} ;
};

extern Token_stream ts;
```

# Functions

```
double expr(bool get){
    double left = term(get);
    for (;;) {
        switch (ts.current().kind) {
            case Kind::plus:
                left += term(true);
                break;
            case Kind::minus:
                left -= term(true);
                break;
            default:
                return left;
        }
    }
}
```

```
double prim(bool get){
    if (get) ts.get();
    switch (ts.current().kind) {
        case Kind::number:
        { double v = ts.current().number_value;
            ts.get();
            return v;
        }
        case Kind::name:
        { double& v = table[ts.current().string_value];
            if (ts.get().kind == Kind::assign) v = expr(true);
            return v;
        }
        case Kind::minus:
            return -prim(true);
        case Kind::lp:
        { auto e = expr(true);
            if (ts.current().kind != Kind::rp) return error("')' expected");
            ts.get();
            return e;
        }
        default:
            return error("primar y expected");
    }
}
```

```
double term(bool get)
{
    double left = prim(get);
    for (;;) {
        switch (ts.current().kind) {
            case Kind::mul:
                left *= prim(true);
                break;
            case Kind::div:
                if (auto d = prim(true)) {
                    left /= d;
                    break;
                }
                return error("divide by 0");
            default:
                return left;
        }
    }
}
```

# Calculate, Parser and Error

```
void calculate()
{
    for (;;) {
        ts.get();
        if (ts.current().kind == Kind::end) break;
        if (ts.current().kind == Kind::print) continue;
        cout << expr(false) << '\n';
    }
}
```

```
int no_of_errors;
double error(const string& s)
{
    no_of_errors++;
    cerr << "error: " << s << '\n';
    return 1;
}
```

```
enum class Kind : char {
    name, number, end, plus='+', minus='-', mul='*', div='/', print=';', assign='=', lp='(', rp=')'
};
struct Token {
    Kind kind;
    string string_value;
    double number_value;
};
```