Notación polonesa inversa:
---

1) estructuración código:

```
string expr;
while (getline(cin, expr)) {
    cout << evaluar(expr) << endl;
}
```

2) cómo tratar la entrada:

i) directamente sobre expr: "154 4 +"
   -- hay que gestionar los espacios en blanco

ii) iss >> c (char c) :
   -- el operador >> se salta los espacios en blanco
   -- problema: no sabemos detectar cuándo acaba y empieza un número

iii) iss >> s (string s):
   -- iss utiliza los espacios en blanco como separadores de string

3) cómo transformar un string de dígitos a valor entero:

i) int str_to_int(const string& s) /* lo implementamos nosotros */

ii) utilizamos alguna función ya implementada:
   -- atoi(s) (-std=c++11)
   -- stoi(s)

4) assert(expresión_booleana);
   -- para comprobar que nuestro código cumple esa condición
   -- al compilar se desactiva con opción -DNDEBUG

Checking parenthesis:
---

1) estructurar código:

i) programa principal:

```
string expr;
while (cin >> expr) {
    if (es_correcta(expr)) cout << ... << endl;
    else cout << ... << endl;
}
```

ii) algunas funciones útiles:

```
bool match(char open, char close) {
        if (open == '(') return close == ')';
        else return close == ']';
}
```

2) recorrido vs. búsqueda:

i) para comprobar la expresión parentizada tenemos que realizar búsqueda:
   -- en el momento en que detectemos error, ya podemos dejar de tratarla.

ii) opciones de implementación (búsqueda):

```
-- utilizando un booleano:

bool correcta = true;
while (correcta and queda_expresión_por_evaluar) {
    if (...) ...
    else if (...) correcta = false;
```

```
        ...
    }
    return correcta ...;

    -- si esa comprobación está encapsulada en una función booleana:

    while (queda_expresión_por_evaluar) {   // bucle puede ser un for
        if (...) ...
        else if (...) return false;
        ...
    }
    return ...;
```

3) diferentes sintaxis for para visitar elementos vector:

```
for (int i = 0; i < expr.size(); ++i) {    ----->   for (char c : expr) {   // char& c
    if (expr[i] ...)                                    if (c ...)
    ...                                                 ...
}                                                   }
```

4) top(), pop(): sólo sobre pilas con elementos

5) estructura del condicional:

```
  if (pila.empty()) return true;
  else return false;
```

simplificar a:

```
  return pila.empty();
```

Simular recursividad (1)

1) significado de la pila: patrones que hay que hacer

2) top de la pila: primer patrón que hay que hacer