Main methods to handle ANTLR ParseTree nodes

ANTLR generates a visitor method for each rule in the grammar. The visitor method looks like:

```
std::any visitRULE(AslParser::RULEContext *ctx)
```

where `RULE` is the name of the rule.

The parameter of each visitor method is a pointer `ctx` to an object of class `RULEContext`, which is derived from ANTLR class `ParseRuleContext`.

The pointer `ctx` allows access to information in the node currently being visited, and to its children.

For instance, if our grammar contains the rule:

```
expr : expr (MUL|DIV) expr
```

then our visitor will have a method:

```
std::any visitExpr(AslParser::ExprContext *ctx)
```

Inside this method we can, for instance:

- Check whether the node has a `MUL` child:

  ```
  if (ctx->MUL()) ...
  ```

- Get the string that matched the `MUL` regex:

  ```
  ctx->MUL()->getText()
  ```

  (not very useful for `MUL`, but very useful e.g. for `ID` or `INTVAL`)

- Check whether the rule has some `expr` child:

```
if (ctx->expr()).../
```

- Get how many `expr` children the rule has:

```
ctx->expr().size()
```

Note: Only when more than one `expr` are expected in the rule/subrule, `ctx->expr()` is a `std::vector<ParseRuleContext>`. Otherwise, `ctx->expr()` is *the* only `expr` child.

- Get the i-th `expr` child (counting from 0):

```
ctx->expr(i)
```

- Get the total number of children of any kind:

```
ctx->children.size()
```

- Get the i-th child of any kind:

```
ctx->children[i]
```

- Both `ctx->expr()` and `ctx->children` can be iterated if needed:

```
for (int i=0; i<ctx->expr().size(); ++i) visit(ctx->expr(i));
for (auto ex : ctx->expr()) visit(ex);

for (int i=0; i<ctx->children.size(); ++i) visit(ctx->children[i]);
for (auto ch : ctx->children) visit(ch);
```

The last two iterations are equivalent to a simple call to `visitChildren(ctx)`.