Expr

Expr — Parent class containing all expression objects available in msdscript

Methods

bool equals()

PTR(Val) interp()

void print()

std::string to_string()

void pretty_print()

void step_interp()

NumExpr

NumExpr – subclass of Expr, integer expression object

AddExpr

AddExpr – subclass of Expr, holds two expression objects being added together

MultExpr

MultExpr – subclass of Expr, holds two expression objects being multiplied together

VarExpr

VarExpr – subclass of Expr, variable expression object

LetExpr

LetExpr – subclass of Expr, enables the use of a defined variable in an expression object

EqExpr

EqExpr – subclass of Expr, equation expression object, used for comparing the equality of two expression objects

BoolExpr

BoolExpr – subclass of Expr, Boolean expression object, holds BoolVal object

IfExpr

IfExpr – subclass of Expr, if expression object used for if, then, else logic.

FunExpr

FunExpr – subclass of Expr, function expression object, contains an “unbound” variable, and a function expression. An unbound variable is a variable that has not been set to a definite value.

CallExpr

CallExpr – subclass of Expr, represents a function call object. Contains a function expression and an argument expression.

Includes

#include "expr.hpp"

`bool equals(PTR(Expr))`

Compares two Expr objects and returns true if they are equal. Two Expr objects are equal if their data types and member fields are equal, otherwise returns false.

Example:

```
PTR(NumExpr) num5 = NEW(NumExpr)(NEW(NumVal)(5));
PTR(NumExpr) number5 = NEW(NumExpr)(NEW(NumVal)(5));
PTR(NumExpr) num4 = NEW(NumExpr)(NEW(NumVal)(4));
PTR(BoolExpr) boolValFalse = NEW(BoolExpr)(false);

std::cout << num5->equals(boolValFalse) << std::endl;
std::cout << num5->equals(num4) << std::endl;
std::cout << num5->equals(number5) << std::endl;

Output: 0, 0, 1
```

`PTR(Val) interp(PTR(Env) env)`

Interprets an Expr and returns its value. The value or Val returned could be a NumVal, a BoolVal, or a FunVal, depending on the expression.

Example:

```
std::string result = (NEW(NumExpr)(NEW(NumVal)(4)))
    ->interp(NEW(EmptyEnv)())->to_string();

std::cout << result << std::endl;

Output: 4
```

void print(std::ostream& out)

This method prints the Expr.

Example:

```
(NEW(AddExpr)(NEW(NumExpr)(NEW(NumVal)(4)),
NEW(NumExpr)(NEW(NumVal)(3))))->print(std::cout);

std::cout << "\n";

Output: (4+3)
```

std::string to_string()

Converts Expr object to a string;

Example:

```
std::string letexpr = (NEW(LetExpr)("x", NEW(NumExpr)(NEW(NumVal)(5)),
NEW(AddExpr)(NEW(LetExpr)("y", NEW(NumExpr)(NEW(NumVal)(3)),
NEW(AddExpr)(NEW(VarExpr)("y"), NEW(NumExpr)(NEW(NumVal)(2)))),
NEW(VarExpr)("x"))))->to_string();

std::cout << letexpr << "\n";

Output: (_let x=5 _in ((_let y=3 _in (y+2))+x))
```

void pretty_print(std::ostream& out)

Converts Expr object to a string;

Example:

```
(NEW(AddExpr)(NEW(NumExpr)(NEW(NumVal)(1)),
NEW(NumExpr)(NEW(NumVal)(2))))->pretty_print(std::cout);

std::cout << "\n";

Output: 1 + 2
```

void step_interp()

This method is used in place of interp() when a user wants to use continuations instead of using stack space. It is the 1st continuation step and must be called from **Step::interp_by_steps ()**.

Val

Val — Parent class containing all value objects available in msdscript

Methods

bool equals()

PTR(Expr) to_expr()

PTR(Val) add_to()

PTR(Val) mult_to()

PTR(Val) call()

std::string to_string()

void call_step()

NumVal

NumVal – subclass of Val, object representing integer values. A NumVal can be added or multiplied. A negative sign will make a NumVal a negative integer value. There is no subtraction in msdscript, to do so, a negative NumVal must be added.

BoolVal

BoolVal – subclass of Val, Boolean value object. Can be true or false.

FunVal

FunVal – subclass of Val, identical to FunExpr expressions except with an additional environment argument used when interpreting function calls.

Includes

#include "val.hpp"

`bool equals(PTR(Val) val)`

Compares two Val objects and returns true if they are equal. Two Val objects are equal if their data types and member fields are equal, otherwise returns false.

Example:

```
PTR(NumVal) num5 = NEW(NumVal)(5);
PTR(NumVal) number5 = NEW(NumVal)(5);
PTR(NumVal) num4 = NEW(NumVal)(4);
PTR(BoolVal) boolValFalse = NEW(BoolVal)(false);

std::cout << num5->equals(boolValFalse) << std::endl;
std::cout << num5->equals(num4) << std::endl;
std::cout << num5->equals(number5) << std::endl;

Output: 0, 0, 1
```

PTR(Expr) to_expr()

Converts a Val object to an Expr object.

Example:

```
#include "expr.hpp"
#include "val.hpp"

std::cout << (NEW(NumVal)(1))->to_expr()->equals(NEW(NumExpr)(NEW(NumVal)(1)))
<< std::endl;

Output: 1
```

PTR(Val) add_to(PTR(Val) rhs)

Adds two NumVal objects.

Example:

```
std::cout << (NEW(NumVal)(4))->add_to(NEW(NumVal)(5))->to_string() <<
std::endl;

Output: 9
```

PTR(Val) mult_to(PTR(Val) rhs)

Multiplies two NumVal objects.

Example:

```
std::cout << (NEW(NumVal)(4))->mult_to(NEW(NumVal)(5))->to_string() <<
std::endl;

Output: 20
```

PTR(Val) call(PTR(Val) actual_arg)

Evaluates function represented in the value object and returns the NumVal result.

Example:

```
PTR(Val) result = (NEW(FunVal)("x", NEW(AddExpr)(NEW(VarExpr)("x"),
NEW(NumExpr)(NEW(NumVal)(2))), NEW(EmptyEnv)()))->call(NEW(NumVal)(10));

std::cout << result->to_string() << std::endl;

Output: 12
```

std::string to_string()

Converts Val object to a string;

Example:

```
std:std::string result = (NEW(FunVal)("x",
NEW(AddExpr)(NEW(VarExpr)("x"),
NEW(NumExpr)(NEW(NumVal)(2))), NEW(EmptyEnv)()))-
>to_string();

std::cout << result << std::endl;

Output: _fun (x) (x+2)
```

void call_step(PTR(Val) actual_arg_val, PTR(Cont) rest)

This method is the 1st continuation step used for funVal objects.

Env

Env — Parent class containing all environment objects in msdscript. An environment represents a set of substitutions to perform. An environment can either be empty (EmptyEnv), or extended (ExtendedEnv).

Methods

PTR(Val) lookup()

EmptyEnv

EmptyEnv – subclass of Env, an empty environment object, meaning there are no substitutions to perform.

ExtendedEnv

ExtendedEnv – subclass of Env, an extended environment object, meaning there are a stack of substitutions to perform.

Includes

#include "env.hpp"

`PTR(Val) lookup(std::string find_name)`

Searches for the variable in the environment. If it exists it will return the Val, usually a NumVal, for that variable.

Example:

```
PTR(Env) env = NEW(ExtendedEnv)("x", NEW(NumVal)(4), NEW(EmptyEnv)());

std::cout << env->lookup("x")->to_string() << std::endl;

Output: 4
```

Step

Step — A class containing static variables and a struct to store information needed for continuations.

Member Variables

typedef enum { interp_mode, continue_mode } mode_t

static mode_t mode

static PTR(Expr) expr

static PTR(Env) env

static PTR(Val) val

static PTR(Cont) cont

Methods

static PTR(Val) interp_by_steps()

Includes

#include "step.hpp"

```
typedef enum {
        interp_mode,
        continue_mode
} mode_t
```

The member struct mode_t has the two possible continuation step modes. A Step::mode can be equal to either interp_mode or continue_mode. There determines whether step_interp() is called or step_continue() in Step::interp_by_steps();

```
static mode_t mode      /* chooses mode */
```

Static member variable for storing the current continuation mode.

```
static PTR(Expr) expr  /* for interp_mode*/
```

Static member variable for storing the current Expr needed for interpreting.

```
static PTR(Env) env    /* for interp mode*/
```

Static member variable storing the current Env needed for interpreting Step::expr.

```
static PTR(Val) val    /* for continue_mode */
```

Static member variable storing the Val received after interpreting Step::expr.

```
static PTR(Cont) cont;  /* all modes */
```
Static member variable storing the current continuation.

```
static PTR(Val) interp_by_steps(PTR(Expr) e);
```
This method is the stepper loop that completes all continuation steps for the Expr object that was passed as an argument. This can be called in place of **Expr::interp()** when there is not enough call stack space to interp() a particular Expr. This is sometimes the case with large number inputs or large numbers that result from a calculation.

Example:

```
Input: _let countdown = _fun(countdown)
                  _fun(n)
                    _if n == 0
                    _then 0
                    _else countdown(countdown)(n + -1)
       _in countdown(countdown)(1000000)


PTR(Expr) expr = Parse::parse_expr(std::cin);
std::cout << expr->interp(NEW(EmptyEnv)())->to_string() << std::endl;


Output: stack-overflow
```

```
Input: _let countdown = _fun(countdown)
                  _fun(n)
                    _if n == 0
                    _then 0
                    _else countdown(countdown)(n + -1)
       _in countdown(countdown)(1000000)


PTR(Expr) expr = Parse::parse_expr(std::cin);
std::cout << Step::interp_by_steps(expr)->to_string() << std::endl;


Output: 0
```

Cont

Cont — Parent class containing all continuation objects in msdscript. Continuation objects remember data needed for continuation steps.

Member Variables

static PTR(Cont) done

Methods

void step_continue()

DoneCont

DoneCont – subclass of Cont, a done continuation object

RightThenAddCont

RightThenAddCont – subclass of Cont, 2nd step continuation object for an AddExpr

AddCont

AddCont – subclass of Cont, last step continuation object for an AddExpr, two expressions are added together.

RightThenMultCont

RightThenMultCont – subclass of Cont, 2nd step continuation object for a MultExpr

MultCont

MultCont – subclass of Cont, last step continuation object for a MultExpr, two expressions are multiplied together

RightThenCompCont

RightThenCompCont – subclass of Cont, 2nd step continuation object for an EqExpr

CompCont

CompCont – subclass of Cont, last step continuation object for an EqExpr, two expressions are compared at this point

IfBranchCont

IfBranchCont – subclass of Cont, continuation object for IfExpr

LetBodyCont

LetBodyCont – subclass of Cont, continuation object for a LetExpr

ArgThenCallCont

ArgThenCallCont – subclass of Cont, 2nd step continuation object for a CallExpr

CallCont

CallCont – subclass of Cont, last step continuation object for a CallExpr

Includes

#include "cont.hpp"

```
static PTR(Cont) done;
```

Static member variable indicating whether a continuation is finished. If Step::cont == Cont::done, Step::interp_by_steps() will return a value, Step::val.

```
void step_continue()
```

This method executes the steps of the continue step for the continuation of all subclasses of Cont.

Parse

Class containing all methods used for parsing user input in msdscript. The input is then converted to the appropriate objects whether they be expressions (Expr) or values (Val).

Member Variables

None.

Methods

- parse_num()
- parse_comparg()
- parse_expr()
- skip_whitespace()
- parse_inner()
- parse_multicand()
- parse_addend()
- parse_let()

Includes

#include "parse.hpp"

```
static PTR(Expr) parse_num(std::istream &in)
```
This method parses integers and creates a corresponding NumExpr object.

```
static PTR(Expr) parse_comparg(std::istream &in)
```

```
static PTR(Expr) parse_expr(std::istream &in)
```
This method parses expressions and creates a corresponding Expr object.

```
static void skip_whitespace(std::istream &in)
```
This method handles white space encountered in user input.

```
static PTR(Expr) parse_inner(std::istream &in)
```

```
static PTR(Expr) parse_multicand(std::istream
&in)
```
This method parses expressions multiplied together and creates a corresponding MultExpr object.

```
static PTR(Expr) parse_addend(std::istream &in)
```

This method parses expressions added together and creates a corresponding AddExpr object.

```
static PTR(Expr) parse_let(std::istream &in)
```

This method parses Let expressions and creates a corresponding LetExpr object.