Homework 1

Due: Tue, Feb 14, 2017

In this assignment, you will be creating an internal language representation for a small expression language.

1 Language

The language has the following kinds of types:

$$t \quad ::= \quad \textbf{bool} \mid \textbf{int}$$

The type **bool** describes the values true ($\top$) and false ($\bot$). The type **int** describes integer values in the left-open range $[-2^{32-1}, 2^{32-1})$.

The language has the following expressions.

e	::=	**true**		$e_1 < e_2$	less than
		false		$e_1 > e_2$	greater than
		n	integer literals	$e_1 \leq e_2$	less than or equal to
		e_1 **and** e_2	and	$e_1 \geq e_2$	greater than or equal to
		e_1 **or** e_2	inclusive or	$e_1 + e_2$	addition
		e_1 **xor** e_2	exclusive or	$e_1 - e_2$	subtraction
		not e_1	logical negation	$e_1 * e_2$	multiplication
		if e_1 **then** e_2 **else** e_3	conditional	e_1 **div** e_2	integer division
		$e_1 = e_2$	equal to	e_1 **rem** e_2	remainder of division
		$e_1 \neq e_2$	not equal to	$-e_1$	arithmetic negation

An expression is a sequence of operands and operators that specifies a computation. The evaluation of an expression results in a value. The type of an expression determines how expressions can be combined to produce complex computations and the kind of value it produces. The following paragraphs define the requirements on operands and the result types of each expression as well the values they produce.

The order in which an expression's operands are evaluated is unspecified unless otherwise noted.

The expressions **true** and **false** have type **bool** and the values $\top$ and $\bot$, respectively.

Integer literals have type **int**. The value of an integer literal is the one indicated by the expression.

The operands of the expressions e_1 **and** e_2, e_1 **or** e_2, and e_1 **xor** e_2 shall have type **bool**. The result type of each is **bool**. The result of e_1 **and** e_2 is $\top$ if both operands are $\top$ and $\bot$ otherwise. The result of e_1 **or** e_2 is $\bot$ if both operands are $\bot$ and $\top$ otherwise. The result of e_1 **xor** e_2 is $\top$ if the operands are different and $\bot$ otherwise.

The operand of **not** e_1 shall have type **bool**, and the type of the expression is **bool**. The result of the expression is $\top$ when the e_1 is $\bot$ and $\bot$ otherwise. Note that **not** e_1 is equivalent to e_1 **xor** 1.

In the expression **if** e_1 **then** e_2 **else** e_3, the type of e_1 shall be **bool**, and e_2 and e_3 shall have the same type. The type of the expression is the type of e_2 and e_3. The result of expression is determined by first evaluating e_1. If that value is $\top$ then the result of the expression is the value of e_2, otherwise, it is the value of e_3. Only one of e_2 or e_3 is evaluated.

The operands of the expressions $e_1 = e_2$ and $e_1 \neq e_2$ shall have the same type. The result type is **bool**. The result of $e_1 = e_2$ is $\top$ if e_1 and e_2 are equal and $\bot$ otherwise. The result of $e_1 \neq e_2$ is $\top$ if e_1 and e_2 are different and $\bot$ otherwise. Note that when e_1 and e_2 have type **bool**, $e_1 \neq e_2$ is equivalent to e_1 **xor** e_2.

The operands of the expressions $e_1 < e_2$, $e_1 > e_2$, $e_1 \leq e_2$, and $e_1 \geq e_2$ shall have type **int**. The result type is **bool**. The result of $e_1 < e_2$ is $\top$ if e_1 is less than e_2 and $\bot$ otherwise. The result of $e_1 > e_2$ is $\top$ if e_1 is greater than e_2 and $\bot$ otherwise. The result of $e_1 \leq e_2$ is $\top$ if e_1 is less than or equal to e_2 and $\bot$ otherwise. The result of $e_1 \geq e_2$ is $\top$ if e_1 is greater than or equal to e_2 and $\bot$ otherwise.

The operands of the expressions $e_1 + e_2$, $e_1 - e_2$, $e_1 * e_2$, e_1 **div** e_2, and e_1 **rem** e_2 shall have type **int**. The result type is **int**. The result of $e_1 + e_2$ is the sum of the operands. If the sum is greater than the maximum value of **int**, the result is undefined. The result of $e_1 + e_2$ is the difference resulting from the subtraction of the e_2 from e_1. If the difference is less than the minimum value of **int**, the result is undefined. The result of $e_1 * e_2$ is the product of the operands. If the product is greater than the maximum value of **int**, the result is undefined. The results of e_1 **div** e_2 and e_1 **rem** e_2 are the quotient and remainder of dividing e_1 by e_2, respectively. In either case, if e_2 is 0, the result is undefined. If e_2 is the minimum value of **int**, the result is undefined. For division, the fractional part of the value is discarded (the value is truncated towards zero). If the expression a **div** b is defined, $(a$ **div** $b) * b + a$ **rem** b is equal to a.

The operand of $-e_1$ shall have type **int**, and the type of the expression is **int**. The result of the expression is the additive inverse of the value of e_1. Note that $-e_1$ is equivalent to $0 - e_1$.

2 Requirements

Implement A C++ framework (collection of classes and functions) that supports this language definition. This must include:

- Implement a set of classes that define the AST for the language. Prefer an object-oriented class hierarchy for the implementation.
- A facility that verifies that an expression is well-type according to the typing rules above.
- A facility that supports the creation of well-typed AST nodes. This should verify the types of operands before constructing the requested node.
- A facility that evaluates an expression (as an AST node).

The type checking and evaluation functions can be implemented as virtual functions or via the visitor design pattern. The tree creation facility does not require virtual functions.

Store your work in an online repository. I recommend GitHub, GitLab, or BitBucket.

Write a short paper to serve as an overview and guide for your implementation. Describe the components of your project in a way that would help a newcomer understand the project's organization. Be sure to include a link to your online source code in your submission.

Submission: Submit your printed homework on the due date. Send me an email with a link to your online source code.

Above and beyond: Make sure that the classes in your hierarchy are well constructed (constructors, access restrictions, appropriate documentation). Note that the semantics of the language could be written

Consider adding two extra expressions:

$$\begin{array}{rcl} e & ::= & \cdots \\ & & e_1 \textsf{ and then } e_2 \\ & & e_1 \textsf{ or else } e_2 \end{array}$$

The expression e_1 **and then** e_2 is equivalent to **if** e_1 **then** e_2 **else false**. The expression e_1 **or else** e_2 is equivalent to **if true then** e_2 **else** e_2.