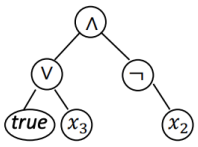
# Assignment6

During lecture you have learned how to create an ROBDD using BUILD in O($2^n$) time. As stated during lecture, a more efficient way is to use APPLY (and MK) and construct the ROBDD bottom-up from the expression tree. The goal of this assignment is to write the pseudo-code for this algorithm.

Consider the expression tree of $(true \vee x_3) \wedge \neg x_2$ shown below




This expression tree and general Boolean expression trees can be represented by the data structure:

| Expr | |
|---|---|
| type() | {VAR, NOT, AND, OR, TRUE, FALSE} |
| left() | Expr |
| right() | Expr |
| idx() | integer |

*Return the type of expression*
*Return the left Expr node of an operation*
*Return the right Expr node of an operation*
*Return the ID of Expr* (*only for type VAR*)

For negated expression (type = NOT), you can assume that right() holds the expression under negation.

## Question 1

1. How can you use APPLY to negate a ROBDD *u*?
   (hint: use one of the 16 Boolean operators and a terminal ROBDD (0 or 1) as your two other arguments to APPLY)

**Answer** So in order to make the negation of the expression $u$ with apply then we can use the $\oplus$, called EXOR(Exclusive OR). So a call to the `Apply` method would look like the following:

```
APPLY(EXOR, 1, U)
```

# Question 2

2. How can you use MK to construct the ROBDD for a variable $x_i$?

**Answer** So in order to use the function `Mk` to construct a ROBDD of a variable $x_i$, we can call it like the following:

```
Mk(i, 0, 1)
```

# Question 3

3. Use your result in 1. and 2. to write the pseudo-code of an algorithm called EXPR2ROBDD that only uses APPLY and MK to create an ROBDD from a Boolean expression of type Expr.

**Answer**

```
function Expr2ROBDD(expr) returns ROBDD

    if expr.type() equals TRUE
        u ← 1
    else if expr.type() equals FALSE
        u ← 0
    else if expr.type() equals VAR
        u ← Mk(expr.idx(), 0 ,1)
    else if expr.type() equals AND
        left = Expr2ROBDD(expr.left())
        right = Expr2ROBDD(expr.right())
        u ← Apply(and,left,right)
    else if expr.type() equals OR
        left = Expr2ROBDD(expr.left())
        right = Expr2ROBDD(expr.right())
        u ← Apply(or,left,right)
    else if expr.type() equals NOT
        right = Expr2ROBDD(expr.right())
        u ← Apply(EXOR,1,right)
    endif

    return u
```