Métodos inicialmente adicionados
-> start_watcher() - inicializa o watcher
-> stop_watcher() - para o watcher
-> ABD_help() - Mostrar protocolo de utilização (eventualmente)
-> regVarChange() - Imprime no ecra, sem qualquer persistencia, que
uma variavel foi alterada e o seu novo valor
########################

PASSOS:

1 - verificar a documentação presente no ficheiro names.c

para adicionar um metodo para ser avaliado à posteriori é necessário um tuplo que contenha na seguinte ordem:

{name, c_function, code, eval, arity, {pp-kind, precedence, rightassoc}}

name: nome a ser usado no R
c_function: o nome da função no código C
Convenção:
- todas começam com "do_"
- todas retornam SEXP
- todas têm a seguinte lista de argumentos (SEXP call, SEXP op, SEXP args, SEXP env)

code: um offset que servirá para distinguir diferentes operaçoes para funçoes R com a mesma funçao C
eval: 3 digitios (XYZ) que definem certas açoes aquando da avaliação (eval)
XYZ:
X = 0 Força R_Visible ON
X = 1 Força R_Visible OFF
X = 2 Permite que o código C actualize conforme desejado
Y = 1 Indica se é uma função interna ou não
Z = 0 Avalia os argumentos antes de chamar (BUILTINSXP)
Z = 1 Não é avaliada (SPECIALSXP)
NOTA: R_Visible é um boleano global que indica se a expressão avaliada deve ser impressa no ecrã após avaliada

arity: argumentos que são necessários (-1 indefinido)

{PP-kind, Precedence, rightassoc} formam um tuplo na última entrada do tuplo base

PP-kind: informação para destinguir a função aquando traduçao
Precedence: Precedencia de operador (avaliar à direita ou à esquerda)
rightassoc: associação à direita ou esquerda

2 - Tuplo {PP-kind, Precedence, rightassoc}

Elemento adicionado ao enum (PPkind) presente no ficheiro Defn.h : PP_WATCHER = 21
Porque? Simplesmente para evitar que sejam efectuadas operações estipuladas para outros PPkinds
Por exemplo: PP_FOR = 7, será usado em certa parte do eval para determinar certas acções para executar um ciclo FOR

Rollback: Foi usado um tuplo (na última entrada do túplo original) equivalente ao presente na definição
da chamada a funções declaradas em R, e nenhuma anomalia foi detectada

Tuplo function: {"function", do_function, 0, 0, -1, {PP_FUNCTION,PREC_FN, 0}}
Conclusão: removido o elemento adicionado no enum (PPkind) presente no ficheiro Defn.h

3 - Tuplos adicionados

```
{"start_watcher",       do_watcher,       1,    100,  0,    {PP_FUNCTION,
     PREC_FN,       0}},
{"stop_watcher",        do_watcher,       2,    100,  0,    {PP_FUNCTION,
     PREC_FN,       0}},
{"ABD_help",            do_watcher,       0,    100,  0,    {PP_FUNCTION,
     PREC_FN,       0}},
```

4 - Criação ficheiro com declarações em C

FILE: ABD_tool_DEF.h
PATH: ../src/include
FINAL_PATH: ../src/include/ABD_tool_DEF.h

CONTENT:

```
void ABD_HELP();
void START_WATCHER();
void STOP_WATCHER();
void regVarChange(SEXP, SEXP);
```

5 - Criação do ficheiro de implementação do ficheiro anterior

FILE: ABD_tool.h
PATH: ../src/include
FINAL_PATH: ../src/include/ABD_tool.h

CONTENT:

```
#include <stdio.h>
#include <Internal.h>
#include <ABD_tool_DEF.h>
static int watcherActivated = 0;

void START_WATCHER(){
    watcherActivated = 1;
}

void STOP_WATCHER(){
    watcherActivated = 0;
}
void ABD_HELP(){

}
void regVarChange(SEXP variable, SEXP newValue){
    if(watcherActivated){
      Rprintf("Some variable modified...\n");
      Rprintf("Variable: %s\n", CHAR(variable));
      Rprintf("New Value: %.0f\n", REAL(newValue)[0]);
    }

}
```

6- Função em chamada aquando da avaliação de: start_watcher(), stop_watcher(), ABD_help()

```
SEXP do_watcher(SEXP call, SEXP op, SEXP args, SEXP rho){
```

```
            switch(PRIMVAL(op)){
                  case 1:
                        // activate_watcher issued
                        START_WATCHER();
                        Rprintf("Watcher STARTED successfully.\n");
                        break;
                  case 2:
                        // stop_watcher issued
                        STOP_WATCHER();
                        Rprintf("Watcher STOPED successfully.\n");
                        break;
                  case 3:
                        // ABD_help() issued
                        ABD_HELP();
                        break;
                  default: errorcall(call, _("invalid usage for ABD_tool.
ABD_help for more info.\n"));
            }
      }
```

7 - adicionar metodo de registo de alteração de variavel

7.1 - Encontrar local "ideal"

Método alvo inicial: consultando a tabela de nomes podemos ver que alterar o valor de variaveis é feito através do método "do_set" presente no ficheiro eval.c

Métodos disponiveis: Dependendo do tipo de atribuição, R faz uso de dois métodos para atribuir o valor a um objecto:

setVar(SEXP symbol, SEXP value, SEXP currEnvironment) para atribuições ao GlobalEnvironment (<<-) através de um while que itera na "árvore" de dependencias

defineVar(SEXP symbol, SEXP value, SEXP currEnvironment) para atribuições ao environment passado por parametro (<-)

Conclusão: defineVar(...) é chamado no final do loop presente em setVar(...) portanto o método onde irá ser feita a modifcação será o defineVar para evitar duplicação de código.

Nota: estes dois metodos (setVar(...) e defineVar(...)) estão implementados no ficheiro envir.c debaixo do caminho ../src/main/

7.2 - "injectar" metodo de registo

O import da biblioteca de headers é essencial (a de implementação não é necessária nesta fase pois já foi importada no ficheiro eval.c e que, é executado e processado antes do envir.c) para que as funçoes estejam disponiveis e que warnings nao sejam lançados.

Após encontrar o local desejado na função, a função para registar a alteração é chamada após R terminar a manipulação para que caso algum erro seja despoletado, a estrutura (que de momento nao existe) que guarda os dados nao contenha informação errónea.