Neste momento a tool e composta por diversos modulos:
- Base
- Object manager
- Event Manager
- Eventos
- Environments Stack Manager
- Json helpers

Base (base.h & base_defn.h)
- No ficheiro base estao os metodos que servem como ponto de entrada a todo o environment que e' a ferramenta ABD.
- Assim sendo o foco deste ficheiro e':
    - Ter metodos que o utilizador pode interagir directamente do R
    - Metodos que os mecanismos do R (que foram modificados) usam para indicar modificacoes quer em objectos, quer em contexto de execucao.

Object manager
- Este Ficheiro tem como proposito:
    - Manter registos para objectos utilizados no programa contendo assim dois registos (cmnObjReg e cfObjReg) formando listas duplamente ligadas (DLL's) do tipo ABD_OBJECT.

    - cmnObjReg – guarda em memoria informacoes referentes a objectos manipulados que não sejam CLOSEXP (closures, por outras palavras, funcoes). Como o nome indica, common object registry, guarda objectos de tipos de dados comuns.
        - Exemplo:
            - a ← 10
            - o object a seria um elemento da DLL cmnRegistry

    - cfObjReg – guarda informacao relativamente a objectos que contenham funcoes guardadas.
        - Exemplo:
            - f1 ← function() { print("123")}
            - f1() //chamada
            - f1 seria guardado na DLL cfObjReg

    - Caracteristica interessante:
        - Sempre que um objecto e' utilizado, ele e' rankeado na lista ou seja:
            - Para cmnObjReg={a,b}
            - se b.createdTime > a.createdTime (b creado depois)
            - e b.usages > a.usages (b tem mais utilizacoes que a)
            - então cmnObjReg={b,a}
        - Ou seja, objectos mais utilizados, serao mais rapidamente encontrados diminuindo, então, o tempo de processamento necessario para os encontrar.
        - Este procedimento acontece tanto no cmnObjReg como no cfObjReg

- ABD_OBJECT & ABD_OBJECT_MOD
    - ABD_OBJECT
        - descreve um objecto para uma identificacao atraves de dados básicos
        - Contem 4 ponteiros
            - 2 ponteiros do tipo ABD_OBJECT_MOD
                - 1 aponta para a primeira modificacao
                - 1 aponta para a ultima modificacao feita

        - 2 ponteiros para ABD_OBJECT
            - 1 para o próximo objecto (OU NULL [cauda da lista])
            - 1 para o objecto anterior (OU NULL [cabeca da lista])
- ABD_OBJECT_MOD
    - 2 ponteiros para ABD_OBJECT_MOD
        - 1 para a próxima modifcacao (OU NULL)
        - 1 para a modificacao anterior (OU NULL)
    - 1 variável do tipo OBJ_STATE
        - pode assumir ADB_ALIVE ou ADB_DEAD
        - para assim determinar se foi uma modificacao de remocao do objecto atraves do metodo rm presente no R
    - 1 variável para guardar o novo valor
        - Ainda não determinado o seu estado final, de momento só um inteiro
    - A lista de modificacoes e' guardada em forma reversa (primeira modificacao aparecera como ultima) para que novas modificacoes sejam adicionadas de imediato a cabeca da lista removendo a necessidade de um ciclo for.
    - (este approach poderia ser usado ao contrario, por ordem natural, apenas teria que se trocar a nomenclatura do ponteiro guardado na raiz do objecto [modListStart → modListTail] e usar esse ponteiro para novas alocacoes).

Event manager

- O conceito do event manager e' que este mantenha um registo de tudo o que aconteceu no programa. Assim sendo ira guardar uma list ligada (LL) sob a variável eventsReg do tipo de dados ABD_EVENT.
- Guardara também um ponteiro para o final desta list (eventsRegTail) que servira para alocacao de novos eventos sem que seja necessario percorrer a lista.
- Guardara ainda uma SEXP denominada de lastRetValue que sera explicada mais a frente.

- ABD_EVENT_TYPE & ABD_EVENT
- ABD_EVENT_TYPE
    - e' um enum que guarda todo o tipo de eventos que a ferramenta ira suportar
    - De momento regista
        - MAIN_EVENT – base da execucao
        - FUNC_EVENT – chamada a uma funcao
        - RET_EVENT – retorno de uma funca
        - IF_EVENT – registara if statements (ainda não feito)
        - (etc..)
- ABD_EVENT
    - 1 variável (type) do tipo ABD_EVENT_TYPE
    - 1 union (data) que ira guardar um ponteiro para cada um dos possíveis tipos de event type (forma de alcançar polimorfismo em C).
        - Exemplo
            - union{
                - ABD_IF_EVENT * if_event;
                - ABD_FUNC_EVENT * func_event;
            - }data;
            - Assim, se a variável type indicar FUNC_EVENT, apenas o ponteiro func_event tera um endereco de memoria, estando os restantes a NULL.
    - 1 ponteiro (nextEvent) para o próximo evento de modo a manter a LL.

- lastRetValue

- Esta variável ira guardar o valor retornado por R quando uma funcao termina a sua execuao. Visto o R trabalhar por partes, isto e, avalia e executa primeiro a funcao e só depois avalia e atribui o retorno a um objecto, uma expressao como a ← f1() torna-se impossivel de fazer em um único sitio (funcao).
- Assim sendo, esta variável ira guardar o ultimo valor retornado por algum objecto do tipo CLOSEXP e, caso haja uma atribuicao em que lastRetValue == objNewValue, então podemos linkar esse objecto ao ret_event da funcao.

Eventos
- ficheiro que descreve e declara todos os tipos de eventos possíveis de serem registados na ferramenta.
- especial atencao para o ABD_FUNC_EVENT
  - 1 ponteiro (objPtr) para um ABD_OBJECT
    - indica que objecto despoletou aquele evento (ex: f1())
  - 1 ponteiro (args) para um ABD_ARGS
    - ABD_ARGS
      - 1 ponteiro (objPtr) para o objecto (ou NULL caso não esteja registado)
      - 1 ponteiro (objValue) para o ABD_OBJECT_MOD
        - com o valor que o objecto tinha na altura
      - 1 ponteiro (nextArg) para o próximo argumento

- ABD_RET_EVENT
  - 1 ponteiro (retValue) do tipo ABD_OBJECT_MOD
    - Este ponteiro ira apontar para um ABD_OBJECT_MOD de um objecto ou sera simplesmente um ABD_OBJECT_MOD não atribuido a nenhum objecto (onde o valor de retorno não foi usado em nenhuma atribuicao)

Environments Stack Manager
- Devido ao comportamento do R (faz varias chamadas [aquando de uma chamada a funcao] que estavam a ser capturadas pela ferramenta e a gerar milhares de entradas lixo) foi necesario desenvolver uma simples stack do tipo ABD_ENV_STACK (sob a forma de uma lista ligada (LL)), denominada envStack, para guardar o environment actual.

- Esses mecanismos que R implementa, e que dificultavam a captura da informacao, são executados mas nunca usando um environment usado pelo utilizador, ou seja, nunca pertence ao GlobalEnvironment ou ao environment da CLOSEXP em execucao.

- ABD_ENV_STACK
  - 1 objecto (rho) do tipo SEXP
  - 1 ponteiro (prev) para o elemento anterior da stack

- funcionamento
  - A stack contem sempre um elemento base (R_GlobalEnv)
  - sempre que uma CLOSXP esteja para ser executada, um newRho (rho → environment) e' criado e temos acesso ao mesmo antes da sua chamada para execucao
  - antes da sua execucao o newRho e' inserido na stack
  - agora sempre que o R fizer chamadas para registar alteracoes num objecto (regVarChange()) ou para registar uma chamada a funcao (regFunCall()), se o rho passado para esses metodos não for o que esta actualmente no topo da stack, essa chamada e' ignorada.
  - Assim, garante-se pleno funcionamento e registo das acoes feitas sobre objectos e funcoes.
  - Sempre que uma funcao retorna, a entrada na stack e' removida e limpa da memoria (free())