In this tutorial, how to call functions that are not written or cannot be written in Alphabets is covered.

Using External Functions

In Alphabets, prototypes of external functions can be declared at the beginning of an Alphabets file. These declared functions can be used like other point-wise operators with variable inputs in Alphabets (MultiArgExpression).

Providing External Functions to the Generated Code

Because the declared functions are assumed to be provided by elsewhere, the code generated with external functions will not compile correctly as is.
When external functions are used, a file named "`external_functions.h`" is also generated in the output directory. This file contains function prototypes that were declared in the alphabets, so that the code can compile. However, compilation of wrapper will still fail, because the function body is not found while linking.

There are two options to get an executable code.

- Modify the make file so that it gets linked with other objects with the body of the function.
- If the function is small, add code to the "`external_functions.h`" created.

Modifying something that was generated need special care, because it can get over-written next time you generate code after minor modification, that do not require change in how external functions are handled.

When the Makefile was modified, the easiest solution is to uncomment the command to generate the Makefile.

When the code is added to the created header, the code generator provides "protected regions" so that the modifications made to the file is preserved.

Example

Let's take a simple example with two external functions. One is a function that computes square root, and another is some user function that returns float given an integer.

```
float sqrt(float);
float userfunc(int);

affine exFuncExample {N|N>0}
given
    float A {i|0<=i<N};
returns
    float C {i|0<=i<N};
through
    C[i] = sqrt(A[i]) + userfunc([i]); //  Note that userfunc is a unary
```

```
pointwise op and so its argument
                                                // must be an alphabets
expression, and we are constructing one with
                                                // the IndexExpr rule of the
grammar.
.
```

Generate code with the following script.

```
prog = ReadAlphabets("exFuncExample.ab");

system = "exFuncExample";

generateWriteC(prog, system, "./"+system);
generateWrapper(prog, system, "./"+system);
generateMakefile(prog, system, "./"+system);
```

Along with the two C codes for `WriteC` and `Wrapper`, you will find a header file, `external_functions.h` with the following contents.

```
//External functions
/***PROTECTED REGION ID(external_functions) ENABLED START***/
float sqrt(float);
float userfunc(int);
/***PROTECTED REGION END***/
```

Code surrounded by PROTECTED REGION is generated when the file or the protected region does not exist, but will be unchanged in subsequent generations.

Now we need to provide implementations for these functions. The first function is actually in the C math library, which is linked by default. Thus we do not need to add body for this function. However, the declaration here will cause conflict with the other declaration of `sqrt`, so this declaration needs to be commented out.

The other function needs a body, and it can be anything as long as it returns a floating point value. An example of the completed header file is shown below.

```
//External functions
/***PROTECTED REGION ID(external_functions) ENABLED START***/
//float sqrt(float);
float userfunc(int in) {
   return -sqrt(in);
}
/***PROTECTED REGION END***/
```

Now try compiling and check if the code works!