

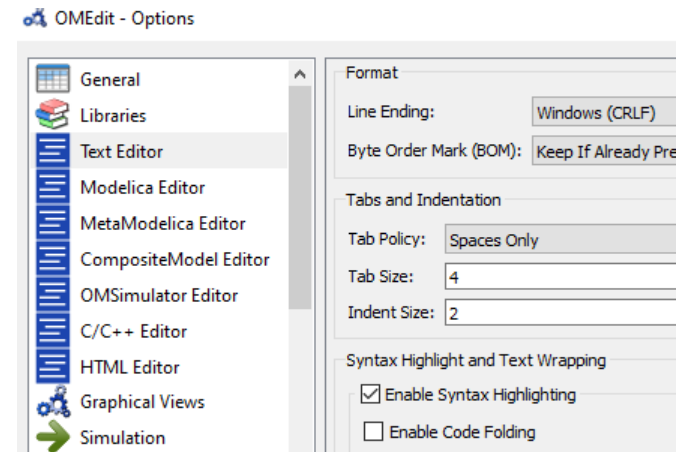
# External Functions – Exercises

- Open ExternalFunctions.mo
- Complete the C definition of function f returning addition of inputs x and y

Hint: add C code line: `return x+y;`

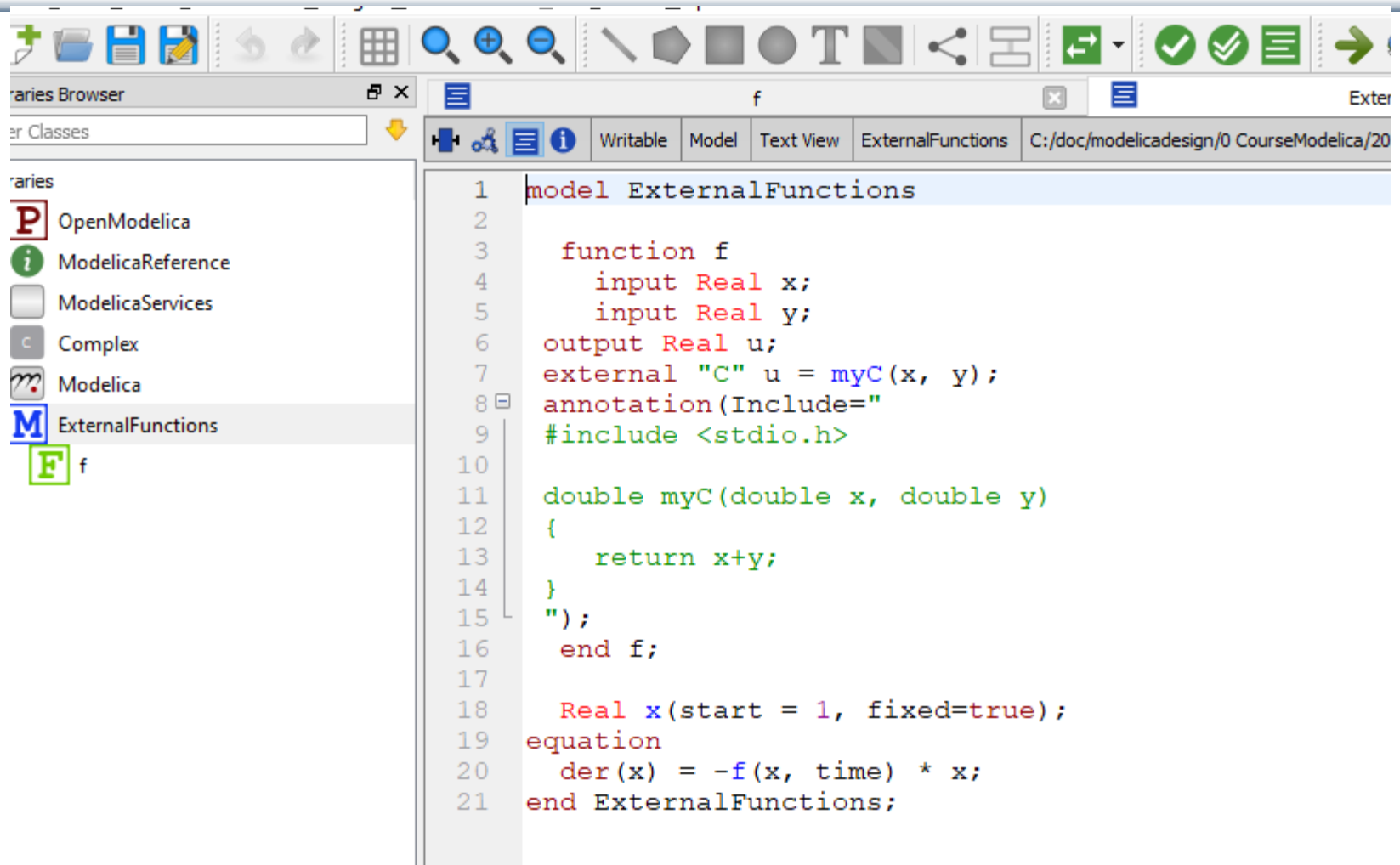
```
7 external "C" u = myC(x, y);  
8 annotation(Include="...");  
16 end f;
```

- Hint: to see the C code click on the + or in Tools->Options disable code folding



- Simulate the model that uses function f
- Optional change the external definition of function f to pass output as argument to the external C function

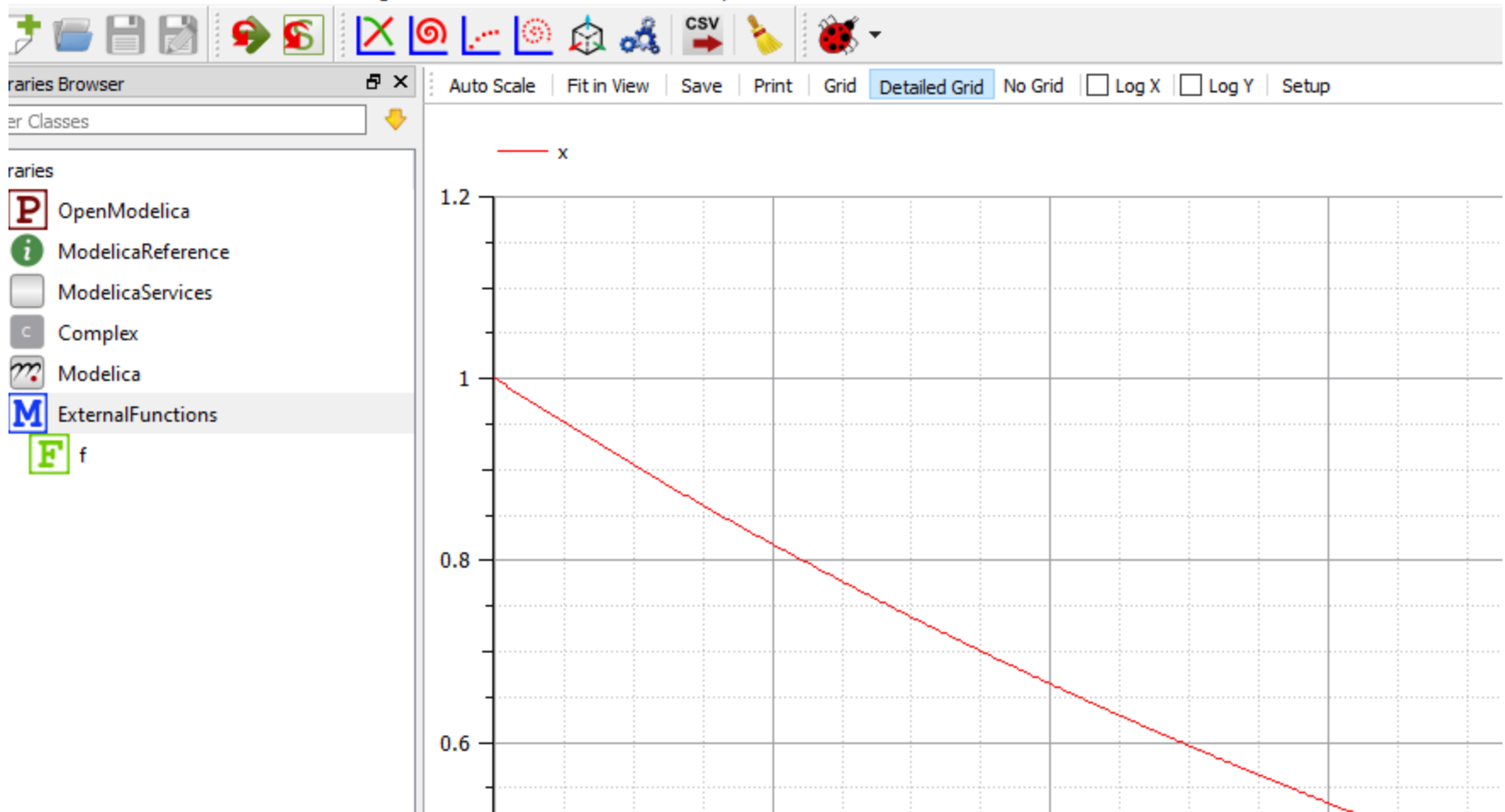
# Solution



The screenshot shows the OpenModelica IDE interface. On the left is the 'Classes Browser' showing a tree of classes: OpenModelica, ModelicaReference, ModelicaServices, Complex, Modelica, ExternalFunctions, and a sub-package 'f' containing a function 'f'. The main editor window displays the source code for the 'f' function, which is defined in C. The code includes a function signature, input/output declarations, an external call to 'myC', an annotation for including 'stdio.h', the implementation of 'myC', and a Modelica equation block that uses the function.

```
1 model ExternalFunctions
2
3   function f
4     input Real x;
5     input Real y;
6   output Real u;
7   external "C" u = myC(x, y);
8   annotation(Include="
9     #include <stdio.h>
10
11   double myC(double x, double y)
12   {
13     return x+y;
14   }
15 ");
16 end f;
17
18 Real x(start = 1, fixed=true);
19 equation
20   der(x) = -f(x, time) * x;
21 end ExternalFunctions;
```

# External function plot



# External Objects

---

- Used for C functions that need to maintain an internal state (opaque to Modelica)
- They are good for making sure things are initialized once (functions can be called multiple time during initialization and simulation). The external object constructor is called only once before the first use of the object

# External Objects – Declaration

- Extend from ExternalObject and provide two functions: constructor and destructor

```
class MyTable
  extends ExternalObject;
function constructor
  input String fileName := "";
  input String tableName := "";
  output MyTable table;
  external "C" table = initMyTable(fileName, tableName);
end constructor;
function destructor "Release storage of table"
  input MyTable table;
  external "C" closeMyTable(table);
end destructor;
end MyTable;
```

# External Objects – Exercises

- Open ExternalObjects.mo
- Fill in the external C code for constructor and destructor for the external object
- Simulate ExternalObjects.Test

```
int i=0; // Hint: you can Insert this code in Constructor
```

```
double *extObj = (double*)malloc(size*sizeof(double));  
if(extObj == NULL)  
    printf("\nNot enough memory\");
```

```
for(i=0; i<size; i++)  
    if(i < 2)  
        extObj[i] = 1.0;  
    else  
        extObj[i] = extObj[i-1]+extObj[i-2];
```

```
return (void*)extObj;
```

Hint, code  
for constructor

Index 0 in C array  
corresponds to index 1  
in Modelica.

```
/* Release storage */  
double *extObj = (double*)object;  
if (object == NULL)  
    return;
```

```
free(extObj);
```

Hint, code  
for destructor

# External objects solution

The screenshot displays the Modelica IDE interface. On the left, the 'Packages Browser' shows a tree of packages: 'OpenModelica', 'ModelicaReference', 'ModelicaServices', 'Complex', 'Modelica', 'ExternalFunctions', 'f', 'ExternalObjects', 'MyExternalObj', 'constructor', 'destructor', 'readFromMyExternalObj', and 'Test'. The 'MyExternalObj' package is selected. The main editor window shows the implementation of 'MyExternalObj' in the 'ExternalObjects' package. The code defines a 'constructor' function that allocates memory for an array of doubles and initializes it with values 1.0 and 2.0. It also defines a 'destructor' function that releases the memory. The code is as follows:

```
18
19     double *extObj = (double*)malloc(size*sizeof(double));
20     if(extObj == NULL)
21         printf("\nNot enough memory\n");
22
23     for(i=0; i<size; i++)
24         if(i < 2)
25             extObj[i] = 1.0;
26         else
27             extObj[i] = extObj[i-1]+extObj[i-2];
28
29     return (void*)extObj;
30
31     });
32 end constructor;
33
34 function destructor
35     input MyExternalObj inMyExternalObj;
36
37     external "C" closeMyExternalObj(inMyExternalObj) ;
38     annotation(Include="
39         #include <stdio.h>
40         #include <stdlib.h> /* for Linux malloc and exit */
41         #include <string.h>
42
43         /* Destructor */
44         void closeMyExternalObj(void *object)
45         {
46             /* release storage */
47             double *extObj = (double*)object;
48             if (object == NULL)
49                 return;
50
51             free(extObj);
```

# Test simulation of External Objects Example

- Test model simulation and plots. Change plot line width via plot Setup.

```
75 model Test
76   parameter Integer size = 5;
77   final parameter Integer size_ = size;
78   parameter MyExternalObj MyExtObj=MyExternalObj(size_);
79   parameter Real p1 = readFromMyExternalObj(MyExtObj, 1);
80   Real p2 = readFromMyExternalObj(MyExtObj, 2);
81   Real p3 = readFromMyExternalObj(MyExtObj, 3);
82   Real p4 = readFromMyExternalObj(MyExtObj, 4);
83   Real p5 = readFromMyExternalObj(MyExtObj, 5);
84 end Test;
```

According to the algorithm  
the first two elements are 1  
The next are the sum of the two  
previous elements

