

## Maple Lecture 9. Code Generation and Connectivity

To study an algorithm, to document its implementation, and investigate its properties, Maple really helps. But often we wish to run the algorithm in a more efficient lower level language, or on a computer where Maple is unavailable. Here is where the code generation facility of Maple may prove useful. On the other hand, we may have a program without a good interface which we want to call from within a Maple session. With a good design of a Maple worksheet interface to the other program, to the user it may seem that the program is part of Maple's library. As repeated calls of a script may not be efficient (especially to run small programs), Maple allows to link directly with C programs. This feature effectively allows us to extend the kernel of Maple, and should be used with care.

In [1, Section 4.6] we find an introduction to code generation with Maple. For more on code generation and to learn more on the interface with other programs, we refer to [2, Chapter 6].

### 9.1 Code Generation

If we wish to export functions, for example to evaluate polynomials or other mathematical expressions, we can generate C or Fortran code. While the newer package `CodeGeneration` is recommended, it does not offer the same capabilities as the older package `codegen`.

```
[> with(codegen);
[> p := x^4 + 3*x + 2;
[> fortran(p, 'optimized', precision=double, mode=double);
[> C(p);
```

To send the output to a file, we proceed as follows:

```
[> C(p, 'optimized', filename="C:\\MCS320\\prog.c");
[> ssystem("type C:\\MCS320\\prog.c");
```

We can also convert Maple procedures into C:

```
[> fp := makeproc(p,x);           # make procedure from formula
[> fp(3);                       # test the function
[> C(fp, 'optimized');
[> fortran(fp, 'optimized');
```

The difference with the previous code generation is that now we get a complete function, instead of just a translation of the code to evaluate the polynomial.

Since Maple 7, there is support for MathML (Mathematical Markup Language), an evolving standard for putting mathematics on the web. Here is how it looks like:

```
[> MathML(fp);
```

Note that the package `codegen` contains several other procedures like `GRAD`, `GRADIENT`, `HESSIAN`, and `JACOBIAN`. These procedures take on input a procedure and return derivatives. Taking derivatives of functions (or programs in general) is known as *automatic differentiation*. We will return to this topic in the third part of this course.

## 9.2 Using scripts

The command `ssystem` allows to launch any program on your computer. For example, to see a directory listing of the C drive, we could create a little alias:

```
[> alias(ls=ssystem("dir c:")):
[> c := ls;
```

Executing a `ssystem` returns what the program normally prints to screen in a string. In the example above, this string is assigned to `c`. In addition, when the program that was called exited normally, `ssystem` returns 0. A nonzero return value indicates an error.

## 9.3 Interfacing with C programs

In this section we illustrate how to have a C program say "Hello world!" to Maple. Our interface worked on a Windows machine using `gcc 2.95`, with Maple 8. What follows is not guaranteed to work on other operating systems, other compilers, and other versions of Maple... Nevertheless, the three steps in preparing, defining and calling the external program will be very similar.

**Step 1: preparing the C program.** The program we wish to call is the following:

```
#include<stdio.h>

char* hello ( void )
{
    char* hi;

    hi = (char*)calloc(12,sizeof(char));
    hi = "hello world!";

    return hi;
}
```

After thoroughly testing this routine, we generate a DLL file, which is a Dynamic Link File, with extension `.dll` on Windows. The relevant instructions in the makefile on our platform are

```
hello:
    c:/mingw32/bin/gcc -c hello.c
    c:/mingw32/bin/gcc -shared -o hello.dll hello.o
```

Typing `make hello` at the command prompt will create the file `hello.dll` needed in the next step.

**Step 2: defining the external link.** We now apply `define_external`

```
[> hi := define_external('hello',C,RETURN::string[],'LIB'="c:/hello.dll");
```

when we place the file `hello.dll` directly on the `c:` drive.

**Step 3: calling the program.** For our application, this is quite straightforward

```
[> hi();
```

but giving in the wrong arguments can bring Maple in a bad state.

An application for this toy example could be the processing in Maple of expressions produced by a C program. Recall that Maple can parse expressions given as strings via the `parse` command.

## 9.4 Passing Data

The program `PrintInteger.c` exemplifies how to pass an integer from a Maple session to a C program. It was prepared on Linux. The C program must include the header file `maplec.h` for the definitions of the Maple types.

```

/* This is a simple example on how to pass an integer from Maple
 * to a C program.  The C program will print the message on the
 * terminal screen where the Maple session was launched.  In the
 * Maple session, the user will see the number returned.
 *
 * Compilation instructions are
 *
 * gcc -shared -I/usr/maple11/extern/include PrintInteger.c \
 *      -L/usr/maple11/bin.IBM_INTEL_LINUX -lmaplec \
 *      -o PrintInteger.so */

#include <stdio.h>
#include <stdlib.h>

#include "maplec.h"

ALGEB PrintInteger ( MKernelVector kv, ALGEB x )
{
    printf("You gave me: %d\n", MapleToInteger32(kv, (ALGEB)x[1]));

    return x;
}

```

The compilation produces a shared object (extension `.so`). In a Maple session, we use `PrintInteger` via

```

> f := define_external("PrintInteger",MAPLE,LIB="PrintInteger.so");
> f(4);

```

## 9.5 Assignments

1. What is the most efficient way to evaluate  $p = x^{32} + x^8 + 1$ ?  
Look at the optimized C code generated by Maple. Count the number of arithmetical operations, running `codegen[cost]` on the output of Maple's generated code.
2. Generate code to evaluate  $p = x^{32} + x^8 + 1$  in any language (C, Fortran, Java, or VisualBasic) for which your computer has a running environment. Choose a couple of random values and report on the comparison in results between the output of the program and the values Maple computes.
3. Use the `makeproc` from the package `codegen` to generate a function from the polynomial  $x^3 - 3x + 2$ . Call the resulting Maple procedure  $p$  and test the outcome by evaluating  $p$  at 23, just by typing  $p(23)$ .
4. Generate a C function to evaluate the "cubic formula" for the *first* root of a general polynomial of degree three:  $ax^3 + bx^2 + cx + d$ , where  $a, b, c$ , and  $d$  are parameters.  
First generate a Maple procedure with arguments  $a, b, c$ , and  $d$  before converting to C.

5. If we type `date/T` in a windows command prompt, then we see the date.

What are the Maple commands you need to be able to type in `today()`; to see today's date?

6. The C compiler we used to demonstrate the interfacing with C programs is available at <http://www.mingw.org>, it is called "MinGW: Minimalist GNU for Windows". Install this compiler (preferably at your home computer, in the labs on campus you will likely not have the right permissions) to create the `hello.dll` file from above. Modify this `hello.c` to make your own message.

Look into the documentation what it takes to have this function take a string on input and print it.

## References

[1] A. Heck. *Introduction to Maple*. Springer-Verlag, third edition, 2003.

[2] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 9 Advanced Programming Guide*. Maplesoft, 2003.