Bergische Universität
Wuppertal

Using PASCAL-XSC Flexible Arrays in C Functions – How to Create an Interface

Andre Weinberg

Preprint 2002/2

Wissenschaftliches Rechnen/
Softwaretechnologie

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster Wissenschaftliches Rechnen/Softwaretechnologie Fachbereich 7 (Mathematik) Bergische Universität Wuppertal Gaußstr. 20 D-42097 Wuppertal

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadresse

Dipl.-Math. Andre Weinberg
Bergische Universität Wuppertal
Gaußstr. 20
D-42097 Wuppertal

E-mail: `weinberg@math.uni-wuppertal.de`

Using PASCAL-XSC Flexible Arrays in C Functions – How to Create an Interface

Andre Weinberg
Department of Mathematics, University of Wuppertal
weinberg@math.uni-wuppertal.de

February 2002

Abstract

In this note we describe a way to use external (e.g. C-) functions within a PASCAL-XSC program to manipulate dynamic and flexible arrays. Using this idea makes it possible to use external libraries in a PASCAL-XSC program or to implement operations not available in PASCAL-XSC like bit operations. Run time comparisons show that this is also a way to create very much faster code for computations which do not need XSC enhancements like interval operations.

1 Indroduction

In standard PASCAL, the size of every structure must be known at compile time. In PASCAL-XSC one can use dynamic and flexible arrays (see [1] for details) in the case that the size of an array is known at run time, not at compile time. To give a dynamic array as an argument to a function, it must be declared as data type:

```
type darray = dynamic array[*] of real;
```

With a function header like

```
function DoSomething ( x : darray; Size : integer ) : SomeType;
```

a variable of a dynamic array of some type can be given as an argument to the function where the size of the array should be given as further argument to be known in the function.

If one wants to use this kind of data structure in an external function written in another programming language (e.g. in C), one has to write an interface between PASCAL-XSC and the other programming language, which works for those dynamic arrays. This may be useful if there is a function or a library available which is written in a programming language like C and it is

not possible or means too much work to translate the source code to PASCAL-XSC. Another intention can be the need for doing bit operations on dynamic arrays, what can not be done in PASCAL-XSC. Also it may be wanted to do some basic work much faster than it is possible with PASCAL-XSC functions.

My intention for solving this problem was a library with some matrix routines for sparse matrices written in C, I wanted to use in a PASCAL-XSC program. When trying to write an interface between my PASCAL-XSC program and the library to use PASCAL-XSC dynamic arrays, there arised a problem: From programmers point of view dynamic arrays are like PASCAL arrays with an improved functionality. But in internal representation they are no arrays. The internal structure contains much more information about the array than just the values.

2 Interface

First we have to mention that PASCAL-XSC does not use standard PASCAL stack order for function or procedure arguments. The stack order of arguments to C functions is used instead. So it is not neccassary to swap the arguments of a function when combining functions written in PASCAL-XSC and in C.

Giving an argument to a (Standard or XSC) PASCAL function with a value that can be changed by the function is practised usually by declaring the argument as `var` parameter. The way to give an argument for a `var` parameter to the function is to put a pointer to the argument variable (the address of the variable) on the stack. This is not a part of the PASCAL standard, but the normally practised implementation of a PASCAL compiler.

The internal representation of dynamic and flexible arrays is not an array, but it contains the array data in an array. Giving the first element of such an array as an argument to a function which has this argument declared as `var` argument is exactly the same as giving the pointer to this first element of the array to the function. That is the way C functions handle arrays resp. pointers in functions. This means that a C function expecting an argument '`int *x`' corresponds to a PASCAL-XSC function with a '`var x : integer`' argument declaration. This kind of function can be called in a PASCAL-XSC statement with the argument `x[1]`, where `x` is an integer array. Finally we have to take care that the data within the array is used in the rigth way. This means that size and order of the used data types have to be the same in both programming languages.

The length of an array can not be determined by the C function, it has to be given as an additional argument.

Integer, real and char arrays

The PASCAL `integer` (32 bit), `real` (64 bit) and `char` (8 bit) types usuallay match the C types `int`, `double` and `char`. So these PASCAL types can be used directly within a C function.

Interval arrays

A PASCAL-XSC `interval` type is a structure (a PASCAL `record`) with two components called `inf` and `sup`. Both components are `real` types. An array of type `interval` can be seen as a `real` array of double size with alternating `inf` and `sup` components. Accessing the data of a PASCAL-XSC `interval` array from a C function can be done in two ways:

- Defining an `interval` structure in C with two `double` components and declaring the C function with an argument '`interval *`'.
- Declaring the C function with a '`double *`' argument and using its even components for the `inf` values and the odd components for the `sup` values (arrays in C start with index 0).

3 Example and tests

The example described here are not very useful in general. It just describes the way to create an interface and is intended to show the idea behind.

What we do here is to use a C function to compute the sum of all components of a PASCAL-XSC real dynamic array. This could all be done within PASCAL-XSC, but this is just an example. We will also do this with a PASCAL-XSC function to compare run time.

First we need the C function:

```
/* vsum.c */
double vsum ( double *v, int n ) {
  int i;
  double sum=0.0, *p;
  for ( i = 0, p = v; i < n; i++, p++ )
    sum += *p;
  return (sum);
  }
```

This C file has to be compiled using the C compiler (and maybe creating very fast code).

Next we need a PASCAL-XSC module in which the (external) function is declared:

```
{ external.p }
module external;
global function vsum ( var v : real ; n : integer ) : real;
                                                external vsum;
begin
  { do nothing here }
end.
```

Now we can look at the main PASCAL-XSC file:

```
program ctest;
use external;
const N = 1000000;
var x : dynamic array[1..N] of real;
    i : integer;
begin
  for i := 1 to N do x[i] := i;
  writeln ('sum is : ', vsum (x[1], N));
end.
```

To link PASCAL-XSC modules with C obejct files, it is helpful to use the `b` command in `dxsc` to create a batch file for compiling and linking. To link your own object files to the PASCAL-XSC object files to get the binary, this batch file can be used by adding the needed objects.

On a SUN workstation running with Solaris the following commands may be used for the example given above:

```
cc -c -fast vsum.c
mxsc external.p
pxsc ctest.p
cc -o ctest ctest.c external.o vsum.o $PXSC_SYS/stdmod.o \
  $PXSC_SYS/rts.a -lm
```

I tested the implementation described above and compared the run time with an implementation in PASCAL-XSC only on a SUN UltraSPARC with 360 MHz. One call of the C function with an array with one million elements took around 0.032 seconds. A PASCAL-XSC function doing the same (computing the sum of one million components) needed 0.496 seconds, which is 15 times slower.

The main reasons for the large difference are the following:

- Combining pointer arithmetics and loop optimization in C produces very fast code.
- Every access to dynamic array components in PASCAL-XSC is preceeded by a bound check, i.e. the array index is compared with the allocated array size.

References

[1] Januschke, P., Ratz, D.: *A Survey of PASCAL-XSC and a Language Reference Supplement on Dynamic and Flexible Arrays*, Bericht, 1/98
http://www.math.uni-wuppertal.de/wrswt/literatur/rep9801.ps.gz