Chapter 3

Mixed Languages interface

In this chapter we describe a mixed languages interface developed for use between the C and FORTH languages. The general ideas and principles used in developing this code are independent of the FORTH and C systems being used. This interface has been compiled under a number of different systems including Borland's "Turbo C", the ZorTech C and C++ compilers in addition to the Microsoft C compiler.

3.1 Principles

The basic principle of the interface is that sufficient state information is stored when switching between FORTH and C operations for both languages to appear to be in full control of the system. The system starts with the C main program which loads and executes the FORTH system. Control will now stay with the FORTH system until such time that FORTH passes control back to C. At this point, C sees the FORTH parameter stack as a simple data structure. The C code will *pop* an item off the FORTH stack and use it as an index into a function table. The requested function is then executed and control is returned to FORTH.

3.2 Argument Passing

All of the argument passing between the two languages is performed by the C system manipulating the FORTH parameter stack as a data structure. Several C functions have been defined to manipulate the FORTH stack. These include operations to *drop* an item, *pop* an item, *push* a value and a function that allows us to *index* into the stack.

In order to make this system more usable, these functions have been defined as type independent macros, thus they take a type indicator as an argument. To pop an integer off the stack we would write the statement "`x = POP(int);`" and to push a floating point value onto the stack the statement "`PUSH(float, n);`" would be used.

3.3 Programming

In order to show the way in which you would program some code, let us look at an interface to the memory allocation system (C heap management).

[0] **This is a chapter taken from the Ph.D. thesis "Practical and Theoretical Aspects of Forth Software Development". A full copy of the thesis is available from the British library. Both the this chapter and the thesis are Copyright © Peter Knaggs.**

```
getmem()
{
  void *ptr;
  int size;

  size = POP(int);
  ptr = (void *)malloc(size);
  PUSH(void *,ptr);
}
```

Figure 3.1: The C `getmem` function.

The code fragment in figure 3.1 is placed in the users C file. A reference to this function must be placed into a jump table (fig 3.2).

```
TBL jmptbl [] =
{
  ...
  /* Function 8 */  getmem,
  ...
}
```

Figure 3.2: Example jump table.

In the FORTH system, a word has been defined that calls the C code via a vector address. To execute this function you would define a FORTH word such as `GETMEM` as shown in figure 3.3.

```
: GETMEM 8 CCALL ;
```

Figure 3.3: The `GETMEM` word.

This would be used as '`1000 GETMEM`'. The C code *pops* the value (1000) into an integer variable (`size`). It will then return a pointer to the memory (`ptr`) allocated by the C system call.

In our implementation, we have defined two FORTH words `CCALL` and `-CCALL` to handle C function calls. Suppose that we wanted to define words to access the function table as given in figure 3.4, we would write the code given in figure 3.5.

The FORTH words `ARC` and `CIRCLE` are defined to call the C code with the relevant function number. However, the `BAR` function is used as a place holder. If at some time in the future we wish to define the `BAR` word, we would simply remove the `-` from the `-CCALL` that is holding `BAR` in place.

3.4 The C Heap

The C system assumes that it has full control of the system memory. Due to this assumption, we must take care when deciding how to load the FORTH system into memory.

The correct method is to request space for the FORTH system from the C heap. FORTH should request memory only via a call to the C system. If the FORTH system were to invoke the memory

```
TBL jmptbl [] =
{
   /* Function 1 */  draw_arc,
   /* Function 2 */  draw_bar,
   /* Function 3 */  draw_circle
}
```

Figure 3.4: Another example jump table

```
 CCALL ARC
-CCALL BAR
 CCALL CIRCLE
```

Figure 3.5: Example of using `CCALL` and `-CCALL` to define words

management system calls directly, this may cause the C heap to become invalid.

This is particularly relevant with regard to MS-DOS where some of the C systems we have used attempt to expand their heap by resizeing the memory space allocated to it rather than by requesting a fresh memory area. If an area of memory allocated to FORTH prevents this operation, the C system will incorrectly assume that it has run out of available memory.

3.5 Organisation

Our system has been split into three modules. The first of these is the main C module that holds the C `main()` function. A second module was written to hold non-portable code (this loads the FORTH system and handles the transfer of control between the two systems). Neither of these modules should be changed by the user. We have placed them into a C library file to be linked in with the third module supplied by the user.

This third module holds all of the users C code and the function jump table. The user compiles this module and links it with the required libraries (including ours) to produce a new C base program.

The source of this interface is far too large to be included here. The full, documented, sources to the interface (including various development macros or scripts), along with a number of technical comments about the system is given, in Appendix C.

3.6 Generalisation

We have made our implementation as general as possible. However, the two routines to load and initialise the FORTH system and to transfer control between the two systems have to be specific to the systems being used.

In our implementation, it is assumed that all memory references are in long (or `far`) form. Thus, it is necessary that all the modules be compiled using the *large* memory model. This is a restriction imposed on use by using a segmented memory structure.

All of the files have been written using standard coding. The FORTH code is a very simple change to the compiled system. The C code has been written to the ANSI standard while the assembler code is written using the standard Microsoft assembler.