Appendix C: Implementation Specifics

This appendix describes implementation-specific details of the iWarp C compiler.

Data types

Table C-1 shows the sizes and alignments of the data types in iWarp C:

Table C-1: Data Type Implementation

Data Type	Size	Alignment	Range	
			min	max
unsigned char	byte (8 bits)	byte	0	+255
char	byte	byte	–128	+127
unsigned short	short (16 bits)	short	0	+65,535
short	short	short	–32768	+32,767
unsigned [int]	word (32 bits)	word	0	+4,294,967,295
int	word	word	–2,147,483,648	+2,147,483,647
unsigned long	word (32 bits)	word	0	+4,294,967,295
long	word	word	–2,147,483,648	+2,147,483,647
float	word	word	–3.402827E+38	+3.402827E+38
double	double (64 bits)	double	[D]	[D]
structure	[A]	double [C]	-	-
union	[A]	double [C]	-	-
arrays	mult of base size	[B] [C]	-	-

[A]: Size rounded up to next multiple of 8 bytes (double)

[B]: Alignment of storage is double unless array size is under 8 bytes, then its alignment is word unless it is either 2 bytes (halfword alignment) or 1 byte (byte alignment).

[C]: malloc(size) always returns double word aligned memory if size >= 8.

[D]: Range is –1.7976931348623157E+308 to +1.7976931348623157E+308.

External Names

The iWarp C compiler implements what is generally called the *Relaxed Ref / Def* model as is normally expected in a UNIX compiler. In C: A Reference Manual, it is referred to as the *Mixed Common Model* (in section 4.8.4). This implementation is recognized as a *common extension* to the C language (see ANSI C Draft Standard A.6.4.11). C programs which take advantage of this model are standard conforming in their environment, but are not maximally portable.

The following is true in the *Relaxed Ref / Def* model for linkage of identifiers:

1. If the `extern` keyword is specified, the declaration is taken to be a reference to a name defined elsewhere. An `extern` declaration cannot have an initializer. If an identifier is declared using `extern` but never used, the compiler will not issue an external reference to the linker This is important, since it means that extra `extern`s will not cause the linker to link in a module that is never used.
2. If the `extern` keyword is *not* specified, and an initializer is present (even zero), a definition for the symbol is emitted. Having two or more such definitions in the same source file will result in a compiler error. Having two or more such definitions among the files which are compiled and linked together will result is a link time error. An initialized definition is still common with identifiers defined elsewhere with `extern` or without either an initializer or an `extern` keyword.
3. If the `extern` keyword is *not* specified, and no initializer is present, a ‘‘common’’ definition, as used in FORTRAN, is emitted. Any number of common definitions of the same identifier may exist, even in the same source file. The ‘‘common’’ definitions are typeless, thus the linker will associate all ‘‘common’’ definitions with the same locations in memory even if one was declared as a ‘float’ and the other as an ‘int’. The compiler does not allow this to occur in a single file and users should not do this in separate files as it is highly non–portable. The linker will not allow different sized ‘‘common’’ definitions to share the same memory, and this will produce a message like: ‘‘Multiply defined symbol *_i*, in *foo.o*, has more than one size”.

<u>C: A Reference Manual</u>, by Samuel P. Harbison and Guy L. Steele Jr., 1987 (second edition), Prentice–Hall.

Limits on sizes

There are hard limits in the compiler which cannot be extended via any command line switch. If you hit one of these limits, divide your program into smaller pieces and compile each separately. The iWarp C compiler has the following limits:

maximum number of formals in a function declaration	300
maximum number of case statements in a switch	1000
maximum nesting of blocks and loop (loop with block counts as 2)	40
maximum number of symbols (symbol table)	4000
maximum size of the table to hold dimension/size information (for arrays, structures, unions, enums) (dimension table)	4500
maximum number of instructions generated per tree by the PCC2 backend	1500
maximum number of nodes generated in a single tree by the front end	1200