

C64 Language Reference

Limitations:

C64 is limited to three dimensional arrays. Arrays of four or more dimensions are not supported.

C64 supports an extended 'C' language compiler. C64 is able to compile most C language programs with little or no modification required. In addition to the standard 'C' language C64 adds the following:

- run-time type identification (via `typenum()`)
- exception handling (via `try/throw/catch`)
- function prolog / epilog control
- multiple case constants eg. case '1','2','3':
- assembler code (`asm`)
- pascal calling conventions (`pascal`)
- no calling conventions (`nocall / naked`)
- additional loop constructs (`until`, `loop`, `forever`)
- `true/false` are defined as 1 and 0 respectively
- thread storage class
- structure alignment control

Compiler Options

Option	Description
-fno-exceptions	This option tells the compiler not to generate code for processing exceptions. It results in smaller code, however the try/catch mechanism will no longer work.
-o	This option disables all optimization done by the compiler causing really poor code to be generated.
-p<processor>	generate code for the specified processor. -pFISA64 for the FISA64 processor
-w	This option disables <code>wchar_t</code> as a keyword. This keyword is sometimes <code>#defined</code> rather than being built into some compilers.

The following additions have been made:

```
typedef(<type>)
```

allow run-time type identification. It returns a hash code for the type specified. It works the same way the sizeof() operator works, but it returns a code for the type, rather than the types size.

C64 supports a simple try/throw/catch mechanism. A catch statement without a variable declaration catches all exceptions.

```
try { <statement> }  
catch(var decl) {  
}  
catch(var decl)  
{  
}  
catch {  
}
```

Types:

A **byte** is one byte (8 bits) in size.

A **char** is two bytes (16 bits) in size.

An **int** is eight bytes (64 bits) wide.

An **short int** is four bytes (32 bits) wide

Pointers are eight bytes (64 bits) wide.

typedef()

Typenum() works like the sizeof() operator, but it returns a hashcode representing the type, rather than the size of the type. Typenum() can be used to identify types at run-time.

```
struct tag { int i; };
```

```
main()  
{  
    int n;  
  
    n = typenum(struct tag);  
}
```

pascal

The pascal keyword causes the compiler to use the pascal calling convention rather than the usual C calling convention. For the pascal calling convention, function arguments are popped off the stack by the called routine. This may allow slightly faster code in some circumstances.

```
pascal char myfunction(int arg1, int arg2)
{
}
```

ncall / naked

The ncall or naked keyword causes the compiler to omit all the conventional stack operations required to call a function. (Omits function prologue and epilogue code) It's use is primarily to allow inline assembler code to handle function calling conventions instead of allowing the compiler to handle the calling convention.

```
ncall myfunction()
{
    asm {
    }
}
```

prolog

The prolog keyword identifies a block of code to be executed as the function prolog. A prolog block may be placed anywhere in a function, but the compiler will output it at the function's entry point.

```
ncall myfunction()
{
    prolog asm {
        // do some prolog work here, eg. setup stack parameters
    }
}
```

epilog

The `epilog` keyword identifies a block of code to be executed as the function epilog code. An epilog block maybe placed anywhere in a function, but the compiler will outout it at the function's return point.

```
nocall myfunction()
{
    // other code
    epilog asm {
        // do some epilog work here, eg. setup return values
    }
}
```

forever

Forever is a loop construct that allows writing an unconditional loop.

```
forever {  
    printf("this prints forever.");  
}
```

case

Case statement may have more than one case constant specified by separating the constants with commas.

C64:

```
switch (option) {  
case 1,2,3,4:  
    printf("option 1-4);  
case 5:  
    printf("option 5");  
}
```

Standard C:

```
switch (option) {  
case 1:  
case 2:  
case 3:  
case 4:  
    printf("option 1-4);  
case 5:  
    printf("option 5");  
}
```

thread

The 'thread' keyword may be applied in variable declarations to indicate that a variable is thread-local. Thread local variables are treated like static declarations by the compiler, except that the variable's storage is allocated in the thread-local-storage segment (tls).

```
thread int varname;
```

align()

The align keyword is used to specify structure alignment in memory. For example the following structure will be aligned on 64 byte boundaries even though the structure itself is smaller in size.

```
struct my_struct align(64) {  
    byte name[40];  
}
```

Place the align keyword just before the opening brace of a structure or union declaration.

Note that specifying the structure alignment overrides the compiler's capability to automatically determine structure alignment. Care must be taken to specify a structure alignment that is at least the size of the structure.

Taking the size of a structure with an alignment specified returns the alignment.