Restricted Pointer Considerations
WG14/N521 (X3J11/95-122)

Bill Homer
Tom MacDonald
Cray Research, Inc.
655F Lone Oak Drive
Eagan, MN 55121
homer@cray.com
tam@cray.com

21 December 1995

Introduction

There are some additional restricted pointer changes we can consider. None of these are essential, but some we may find to be useful.

The first proposed change is to add the **restrict** keyword to some library functions when copying between overlapping objects might introduce undefined behavior. At the moment, several library functions do not say anything about copying between overlapping objects even though they introduce undefined behavior.

The second change is to allow type qualifiers (like **restrict**) when declaring a parameter to have an array type, which is then automatically adjusted to be a pointer. This is especially convenient for the **restrict** keyword.

The last proposed change allows the programmer to declare a pointer to a constant value (as opposed to a read-only value). This involves a combination of both the **restrict** and **const** keywords.

Library Changes

Several library functions have parameters that could be adorned with the **restrict** keyword. These library functions are listed below.

```
#include <stdio.h>
int fprintf(FILE * restrict stream, const char * restrict format, ...);
int fscanf(FILE * restrict stream, const char * restrict format, ...);
int printf(const char * restrict format, ...);
int scanf(const char * restrict format, ...);
int sprintf(char * restrict s, const char * restrict format, ...);
int sscanf(char * restrict s, const char * restrict format, ...);
#include <stdarg.h>
int vfprintf(FILE * restrict stream, const char * restrict format, va_list arg);
int vprintf(const char * restrict format, va_list arg);
int vsprintf(char * restrict s, const char * restrict format, va_list arg);
char *fgets(char * restrict s, int n, FILE * restrict stream);
char *fputs(const char * restrict s, FILE * restrict stream);
```

2 11

```
size_t fread(void * restrict ptr, size_t size, size_t nmemb,
        FILE * restrict stream);
size_t fwrite(void * restrict ptr, size_t size, size_t nmemb,
        FILE * restrict stream);
int fgetpos(FILE * restrict stream, fpos_t * restrict pos);
#include <stdlib>
int mbtowc(wchar_t * restrict pwc, const char * restrict s, size_t n);
size_t mbstowcs(wchar_t * restrict pwcs, const char * restrict s, size_t n);
size_t wcstombs(char * restrict s, const wchar_t * restrict pwcs, size_t n);
#include <string.h>
void *memcpy(void * restrict s1, const void * restrict s2, size_t n);
char *strcpy(char * restrict s1, const char * restrict s2);
char *strncpy(char * restrict s1, const char * restrict s2, size_t n);
char *strcat(char * restrict s1, const char * restrict s2);
char *strncat(char * restrict s1, const char * restrict s2, size_t n);
size_t strxfrm(char * restrict s1, const char * restrict s2, size_t n);
char *strtok(char * restrict s1, const char * restrict s2);
size_t strftime(char * restrict s, size_t maxsize,
        const char * restrict format, const struct tm * restrict timeptr);
#include <wchar.h>
int fwprintf(FILE * restrict stream, const wchar_t * restrict format, ...);
int fwscanf(FILE * restrict stream, const wchar_t * restrict format, ...);
int wprintf(const wchar_t * restrict format, ...);
int wscanf(const wchar_t * restrict format, ...);
int swprintf(wchar_t * restrict s, const wchar_t * restrict format, ...);
int swscanf(wchar_t * restrict s, const wchar_t * restrict format, ...);
int vfwprintf(FILE * restrict stream, const wchar_t * restrict format,
        va_list arg);
int vwprintf(const wchar_t * restrict format, va_list arg);
int vswprintf(wchar_t * restrict s, size_t n, const wchar_t * restrict format,
        va_list arg);
char *fgetws(wchar_t * restrict s, int n, FILE * restrict stream);
char *fputws(const wchar_t * restrict s, FILE * restrict stream);
char *wcscpy(wchar_t * restrict s1, const wchar_t * restrict s2);
char *wcsncpy(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n);
char *wcscat(wchar_t * restrict s1, const wchar_t * restrict s2);
char *wcsncat(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n);
size_t strxfrm(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n);
wchar_t *wcstok(wchar_t * restrict s1, const wchar_t * restrict s2,
        wchar_t * restrict * restrict ptr););
void *memcpy(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n);
size_t strftime(wchar_t * restrict s, size_t maxsize,
        const wchar_t * restrict format, const struct tm * restrict timeptr);
size_t mbrlen(const char * restrict s, size_t n, mbstate_t * restrict ps);
size_t mbrtowc (wchar_t * restrict pwc, const char * restrict s, size_t n,
        mbstate_t * restrict ps);
size_t wcrtomb(char * restrict s, wchar_t wc, mbstate_t * restrict ps);
size_t mbstrowcs(wchar_t * restrict dst, const char * restrict * restrict src,
        size_t len, mbstate_t *restrict ps);
size_t wcsrtombs(char * restrict dst, const wchar_t * restrict * restrict src,
        size_t len, mbstate_t * restrict ps);
```

In all cases except **wcstok**, **mbstrowcs**, and **wcsrtombs** the qualifiers are on outermost pointer type derivations, and so the proposed declaration is compatible with the current declaration.

2/2

We may also want to say that if the type **`va_list`** in the header **`<stdarg.h>`** is a pointer type, then it shall be a restrict-qualified pointer type. This eliminates possible aliasing problems with **`vfprintf`** (for example).

Array Parameters

By ISO section 6.7.1, a declaration of a function parameter as "array of **`type`**" is adjusted to "pointer to **`type`**." The following changes provide a means of expressing qualifiers for the adjusted type.

In ISO section 6.5.4, Declarators, in subsection Syntax, allow an optional *type-qualifier-list* in the third form of direct-declarator:

```
direct-declarator [ type-qualifier-list_opt constant-expression_opt ]
```

Note that the VLA proposal also modifies this same line by allowing expressions that are not constant.

In ISO section 6.5.4.2, Array Declarators, in subsection Constraints, add: **Type qualifiers shall appear preceding or in place of a size expression only in a declaration of a function parameter of array type, and then only in the outermost array type derivation.**

In ISO section 6.7.1, Function Definition, under subsection Semantics, modify lines 23-24 to read: **A declaration of a parameter as "array of *type*" shall be adjusted to "qualified pointer to *type*" (where the type qualifiers are those specified within the square brackets of the array type derivation), ...**

Rationale

This extension is especially convenient for using the **`restrict`** qualifier with variable length array parameters, as shown in the following example. The extension makes the first declaration of **`f`** conforming, and equivalent to the second. The first declaration both documents the role of parameter **`m`**, and more clearly conveys the fact that parameters **`a`** and **`b`** refer to two dimensional arrays.

Example 1

```
void f(int m, int n, float a[restrict m][n],
                     float b[restrict m][n]);

void f(int m, int n, float (* restrict a)[n],
                     float (* restrict b)[n]);
```

Restricted Pointer to Const

Add to the semantics of restricted pointers the new third sentence in the following paragraph:

During each execution of **`B`**, let **`O`** be the array object that is determined dynamically by all references through pointer expressions based on **`P`**. Then *all* references to values of **`O`** shall be through pointer expressions based on **`P`**. **If P was designated as a restrict-qualified pointer to a const-qualified type, then none of the references shall modify the value of O.** Furthermore, if **`P`** is assigned the value of a pointer expression **`E`** that is based on another restricted pointer object **`2`** associated with block **`B2`**, then either the execution of **`B2`** shall begin before the execution of **`B`**, or the execution of **`B2`** shall end prior to the assignment. If these requirements are not met, then the behavior is undefined.

213

Rationale

This change enables a declaration to assert that an object referred to through a pointer has a constant (i.e., unmodified) value. This is especially useful for dynamically allocated data structures.

Without this change, a translator cannot deduce from presence of **const** in the declaration of **x** below that the value of the associated object (or array) does not change during the execution of **g**.

Example 2

```
void g(const float * restrict x /*, ... */ )
{
    /* Without the change, const does not mean constant: */
    float *y = (float *)x;
    *y = 0.0;
    /* The change renders this function non-conforming. */
}
```