Type generic string interfaces
honor the const contract of application code

Jens Gustedt
INRIA and ICube, Université de Strasbourg, France

In several places, C library functions break the const contract of user code by returning an unqualified pointer to a const qualified object. This situation arises because these library functions are meant to deal with both, const qualified and unqualified pointer targets. With C11’s _Generic such a break of const can easily avoided and it can be used to provide type generic interfaces that are const correct. As additional fall out, such type generic interfaces can combine the functionality of narrow and wide character support functions.

Introduction
The following string search functions in <string.h> are potentially dangerous, since they break the const contract of application code. They return a pointer that is not const-qualified which is pointing to a const qualified object that is received as a pointer parameter:

```c
void *memchr(void const *s, int c, size_t n);
char *strchr(char const *s, int c);
char *strrchr(char const *s1, const char *s2);
char *strchr(char const *s, int c);
char *strstr(char const *s1, const char *s2);
```

If the application code stores the returned pointer in a variable and uses this later this can lead to two types of problems:

— Writing to read only memory. The result would be a runtime crash of the program.
— A subtle unnoticed modification of an object that is received through a const-qualified pointer, and thus a break of an interface contract.

In any case writing to such an object has undefined behavior.

Tracking provenance of pointers to objects to remember if an object is writable or not is a tedious enterprise. const-qualification had been introduced to C to avoid the need of such provenance tracking and C library functions that break this contract are jeopardizing the whole idea.

**GOAL 1. Ensure that all C library interfaces honor the const contract.**

As a simple fall out of using type generic functions for str* and mem* functions is that we also can use these to unify functions for char* and wchar_t, much as <tgmath.h> provides type generic interfaces for the functions in <math.h>.

**GOAL 2. Provide type generic interfaces for narrow and wide string functions.**

The string-to-number conversion functions from <stdlib.h> and <inttypes.h> such as

```c
double strtod(const char * restrict nptr, char ** restrict endptr);
```

exhibit an even worse problem, because endptr is a pointer to pointer that is forcibly the wrong type if the origin of nptr had been const qualified: if endptr is non-null and the string is successfully scanned, the function stores nptr+x (of type const char*) into the pointer *endptr (of type char*). If the const properties of endptr are not correctly tracked by the application code, the application can be tempted to write into a const qualified object and thereby encounter undefined behavior.
Previous versions of the C standard followed that strategy for the interface specification because they wanted to ensure that these functions can be called for both types of strings, those that are `const` qualified and those that are or not. This was necessary because many of these functions existed before `const` was even integrated into C, backwards compatibility was a prime objective, and adding qualifiers to pointer targets was the only way to allow for qualified and unqualified arguments.

This situation has changed with C11. Because it added type generic selection with the keyword `__Generic`, type generic interfaces can now easily be added to the standard and help applications to keep their `const` contracts more easily.

In C++, from which C inherited `const` qualification, such functions would be overloaded with different interfaces that distinguish according to the `const`-qualification of their argument.

```c
#ifdef __cplusplus
void *memchr(void *, int, size_t);
void const*memchr(void const *, int, size_t);
char *strchr(char *, int);
char const*strchr(char const *, int);
char *strrchr(char *, int);
char const*strrchr(char const *, int);
char *strstr(char *, char *);
char const*strstr(char const *, char *);
double strtod(char *, char **);
double strtod(const char *, char const**);
#endif
```

### A solution for C using `__Generic`

With `__Generic` we now have the possibility to have a similar interface for C. We can provide macros of the same name as the standard library functions and provide the semantics of function overloading.

At the same time that we overcome the `const` qualification problem, we may also simplify the use of string functions even more. Most string functions and byte functions are copied by interfaces for wide character strings, that have analogous interfaces with `wchar_t` instead of `char`. For them, equivalent interfaces can be provided that also map the functionalities to the corresponding `str` names.

The following macro selects one of seven function pointers according to the first argument being

- a `void` pointer,
- a narrow character pointer, or
- a wide character pointer

and according to `const` qualification.

```c
/**
 * @brief An internal macro to select a const conserving string
 */
```
** function.
** This macro simply selects the function that corresponds to
** the type of argument @a X. It is not meant to be used
** directly but to be integrated in specific macros that
** overload C library functions.
**
** The argument @a X must correspond to an array or pointer,
** otherwise an error is triggered in the controlling
** expression.
**
** If any of the 7 cases makes no sense for the type generic
** interface in question, that case can be switched off by
** passing a 0 instead of a function. A use of that case in user
** code then results in a compile time error.
**
** @see memchr on how to integrate this into a type generic user
** interface
**
/**
#define _TG_TO_CONST(DF, VN, VC, SN, SC, WN, WC, X, ...) \
_Generic(&(X[0]), \
  default: DF, \
  void*: VN, \
  void const*: VC, \
  char*: SN, \
  char const*: SC, \
  wchar_t*: WN, \
  wchar_t const*: WC \
)((X), _V A R G S _ _ )

Then we add auxiliary definitions such as the following two:

void const*memchr(const(void const* _s,
       int _c, size_t _n) {
  return memchr(_s, _c, _n);
}

wchar_t const*wmemchr(const(wchar_t const _s[static 1],
       wchar_t _c, size_t _n) {
  return wmemchr(_s, _c, _n);
}

Here, these functions can be of some specialized storage class, or attribute such as static
inline, register or _attribute(_Always_inline) if such features are included in C2x.

Then we define memchr as a macro:

#undef memchr
#define memchr(...) \
_TG_TO_CONST(memchr, /* default */ \
memchr, /* void* */ \
memchr const, /* const void* */ \

memchr, /* char* */ \nmemchr_const, /* const char* */ \nwmemchr, /* wchar_t* */ \nwmemchr_const, /* const wchar_t* */ \n_VA_ARGS_)

In this case of a “mem” function a default fallback makes sense. Any pointer type should still fallback to the void* variant.

In contrast to that “str” functions suppose that their arguments point to strings, that is to null terminated character arrays. With analogous definitions, for strstr we can force errors for the usage for the first three cases:

```
# undef strrchr
# define strrchr(...) \
_TG_TO_CONST(0, 0, 0, /* the first three cases are errors */ \
strchr, strchr_const,wcschr, wcschr_const, \
_VA_ARGS_)
```

The functions that should be covered by such type generic macros are: memchr, memcpy, memmove, memset, strlen, strncat, strncmp, strnep, strpbrk, strcch, strspn, strstr, strtod, strtol, strtof, strtok, strtol, strtoll, strtof, strtoull, strtoimax, wmemchr, wcsncmp, wmemcmp, wmemmove, wcschr, wcsspn, wcsstr, wcstod, wcstof, wcsstr, wcstol, wcsch, wcstoul, wcsch, wcschr, wmemchr, wmemcmp, wmemmove, and wmemset.

Appendix: a reference implementation

We provide a reference implementation of the proposed header that is intended to be usable with any compliant C compiler and that doesn’t need any addition to an existing C library. The license for this is free, so it can be directly used by C library implementations.

```
#ifndef __STDC_TGSTRING__
#define __STDC_TGSTRING__ 201610L

/**
 * @file
 * @brief Type generic string interfaces
 *
 * @copyright 2016 Jens Gustedt
 *
 * @copy
 */

/**
 * If __STDC_VERSION__ < 201112L
 * error "tgstring.h needs a C11 conforming compiler"
 */

/* We define a bunch of inline functions that have the only
The purpose of providing conversions to the their arguments and then to pass them on to the corresponding standard function. We use the same names as the standard function and append "const" to them. This use is covered by the standard since all of the "str" and "wcs" names are reserved, anyhow. */

/* If the register extension is included in the standard, this would be an ideal candidate. */
#if __STDC_REGISTER_IN_FUNCTION_SCOPE__
#define _TG_INLINE register
#else
// void *memchr(void const *s, int c, size_t n);
_TG_INLINE
void const*memchr_const(void const* __s, int __c, size_t __n) {
return memchr(__s, __c, __n);
}
#endif
# undef memchr
#define memchr(...) 
   _TG_TO_CONST(memchr, /* default */
                 memchr, /* void* */
                 memchr_const, /* const void* */
                 memchr, /* char* */
                 memchr_const, /* const char* */
                 wmemchr, /* wchar_t* */
                 wmemchr_const, /* const wchar_t* */
                 _ _ V A _ A R G S _ _)

// char *strchr(char const *s, int c);

# undef strchr
#define strchr(...) 
   _TG_TO_CONST(0, 0, 0, 
                 strchr, strchr_const, wcschr, wcschr_const, 
                 _ _ V A _ A R G S _ _)

// char *strrchr(char const *s, int c);

# undef strpbrk
#define strpbrk(...) 
   _TG_TO_CONST(0, 0, 0, 
                 strpbrk, strpbrk_const, wcnstrpbrk, wcnstrpbrk_const, 
                 _ _ V A _ A R G S _ _)

// char *strrchr(char const *s, int c);
char const* strrchr_const(char const *s, int c) {
    return strrchr(s, c);
}

wchar_t const* wcsrchr_const(wchar_t const *s, wchar_t _c) {
    return wcsrchr(s, _c);
}

#undef strrchr
#define strrchr(...) 
    _TG_TO_CONST(0, 0, 0, /* the first three cases are errors */ 
    strrchr, strrchr_const, wcsrchr, wcsrchr_const, 
    __VA_ARGS__)

// char *strstr(char const *s, const char *t);

char const* strstr_const(char const *s, char const *t) {
    return strstr(s, t);
}

wchar_t const* wcsstr_const(wchar_t const *s, wchar_t const *t) {
    return wcsstr(s, t);
}

#undef strstr
#define strstr(...) 
    _TG_TO_CONST(0, /* default type: error */ 
    0, /* void*: error */ 
    0, /* const*: error */ 
    strstr, strstr_const, wcsstr, wcsstr_const, 
    __VA_ARGS__)

/**
 * @brief An internal macro to select a memory function
 * according to the base type @c void* or @c wchar_t*.
 * This macro simply adds the "w" prefix to the name of the
 * function, if a @c wchar_t function is to be called. All other
 * cases are mapped to the @c void*.
 * The argument @a X must correspond to an array or pointer,
 * otherwise an error is triggered in the controlling
 * expression.
 * @see memcpy on how to integrate this into a type generic user
 * interface
 */
#define _TG_MEM(NAME, X, ...) 
    _Generic(&(X[0]), 
    0, /* default type: error */ 
    0, /* void*: error */ 
    0, /* const*: error */ 
    strstr, strstr_const, wcsstr, wcsstr_const, 
    __VA_ARGS__)

/**
 * @brief An internal macro to select a memory function
 * according to the base type @c void* or @c wchar_t*.
 * This macro simply adds the "w" prefix to the name of the
 * function, if a @c wchar_t function is to be called. All other
 * cases are mapped to the @c void*.
 * The argument @a X must correspond to an array or pointer,
 * otherwise an error is triggered in the controlling
 * expression.
 * @see memcpy on how to integrate this into a type generic user
 * interface
 */
#define _TG_MEM(NAME, X, ...) 
    _Generic(&(X[0]), 
    0, /* default type: error */ 
    0, /* void*: error */ 
    0, /* const*: error */ 
    strstr, strstr_const, wcsstr, wcsstr_const, 
    __VA_ARGS__)

/**
 * @brief An internal macro to select a memory function
 * according to the base type @c void* or @c wchar_t*.
 * This macro simply adds the "w" prefix to the name of the
 * function, if a @c wchar_t function is to be called. All other
 * cases are mapped to the @c void*.
 * The argument @a X must correspond to an array or pointer,
 * otherwise an error is triggered in the controlling
 * expression.
 * @see memcpy on how to integrate this into a type generic user
 * interface
 */
#define _TG_MEM(NAME, X, ...) 
    _Generic(&(X[0]), 
    0, /* default type: error */ 
    0, /* void*: error */ 
    0, /* const*: error */ 
    strstr, strstr_const, wcsstr, wcsstr_const, 
    __VA_ARGS__)

/**
 * @brief An internal macro to select a memory function
 * according to the base type @c void* or @c wchar_t*.
 * This macro simply adds the "w" prefix to the name of the
 * function, if a @c wchar_t function is to be called. All other
 * cases are mapped to the @c void*.
 * The argument @a X must correspond to an array or pointer,
 * otherwise an error is triggered in the controlling
 * expression.
 * @see memcpy on how to integrate this into a type generic user
 * interface
 */
#define _TG_MEM(NAME, X, ...) 
    _Generic(&(X[0]), 
    0, /* default type: error */ 
    0, /* void*: error */ 
    0, /* const*: error */ 
    strstr, strstr_const, wcsstr, wcsstr_const, 
    __VA_ARGS__)

/**
 * @brief An internal macro to select a memory function
 * according to the base type @c void* or @c wchar_t*.
 * This macro simply adds the "w" prefix to the name of the
 * function, if a @c wchar_t function is to be called. All other
 * cases are mapped to the @c void*.
 * The argument @a X must correspond to an array or pointer,
 * otherwise an error is triggered in the controlling
 * expression.
 * @see memcpy on how to integrate this into a type generic user
 * interface
 */
#define _TG_MEM(NAME, X, ...) 
    _Generic(&(X[0]), 
    0, /* default type: error */ 
    0, /* void*: error */ 
    0, /* const*: error */ 
    strstr, strstr_const, wcsstr, wcsstr_const, 
    __VA_ARGS__)

/**
 * @brief An internal macro to select a memory function
 * according to the base type @c void* or @c wchar_t*.
 * This macro simply adds the "w" prefix to the name of the
 * function, if a @c wchar_t function is to be called. All other
 * cases are mapped to the @c void*.
 * The argument @a X must correspond to an array or pointer,
# undef memcpy
# define memcpy(...) _TG_MEM(memcpy, __VA_ARGS__)
# undef memmove
# define memmove(...) _TG_MEM(memmove, __VA_ARGS__)
# undef memcmp
# define memcmp(...) _TG_MEM(memcmp, __VA_ARGS__)
# undef memset
# define memset(...) _TG_MEM(memset, __VA_ARGS__)

/**
 * @brief An internal macro to select a string function
 * according to the base type @c char or @c wchar_t.
 * This macro simply adds the "wcs" prefix to the name of the
 * function, if a @c wchar_t function is to be called. Traditional @c char strings are mapped to @c char.
 * There is no default case, because these functions need a
 * string and not some arbitrary bytes.
 * The argument @a X must correspond to an array or pointer,
 * otherwise an error is triggered in the controlling
 * expression.
 * No precaution is made concerning @c const. If a @c const qualified pointer type is passed in and the function that is
 * selected does not accept such an argument, an error is
 * diagnosed as if the corresponding function would have been
 * used directly.
 * @see strcpy on how to integrate this into a type generic user
 * interface
 */
#define _TG_STR(NAME, X, ...) 
_Generic(&(X[0]),
     char*: str ## NAME, 
     char const*: str ## NAME, 
     wchar_t*: wcs ## NAME, 
     wchar_t const*: wcs ## NAME 
     )((X), __VA_ARGS__)

#undef strcpy
#define strcpy(...) _TG_STR(cpy, __VA_ARGS__)
#undef strncpy
#define strncpy(...) _TG_STR(ncpy, __VA_ARGS__)
#undef strcat
#define strcat(...) _TG_STR(cat, __VA_ARGS__)
#undef strncat
#define strncat(...) _TG_STR(ncat, __VA_ARGS__)
#undef strcmp
#define strcmp(...) _TG_STR(cmp, __VA_ARGS__)
#undef strcoll
### Type generic string interfaces

```c
#define strcoll(...) _TG_STR(coll, __VA_ARGS__)
#undef strncmp
#define strncmp(...) _TG_STR(ncmp, __VA_ARGS__)
#undef strcspn
#define strcspn(...) _TG_STR(cspn, __VA_ARGS__)
#undef strspn
#define strspn(...) _TG_STR(spn, __VA_ARGS__)
#undef strntok
#define strntok(...) _TG_STR(ntok, __VA_ARGS__)
#undef strlen
#define strlen(...) _TG_STR(len, __VA_ARGS__)

/**
 * @brief An internal Xmacro to generate const conserving string
 * to floating point conversion function.
 * In the case of the string to number functions the original C
 * library functions cannot be used directly, because their
 * second argument is an unqualified pointer to pointer.
 *
 * @param T is the target type of the conversion
 * @param C is the base type of the string, so @c char or @c wchar_t
 * @param NAME is the base name for the function
 **
#define _TG_STR_F_FUNC_(T, C, NAME) \
  _TG_INLINE \n  T _TG_ ## NAME ## _c(C const* _p, C const** _e) { \n    return NAME(_p, (C**)_e); \n  } \n  _TG_INLINE \n  T _TG_ ## NAME ## _nc(C* _p, C** _e) { \n    return NAME(_p, _e); \n  } \n  typedef T _TG_ ## NAME ## _c_ftype(C const*, C const**); \n  typedef T _TG_ ## NAME ## _nc_ftype(C*, C**)

/**
 * @brief An internal Xmacro to generate const conserving string
 * to floating point conversion function.
 * This macro just assembles _TG_STR_F_FUNC_() for the two case
 * of @c C being @c char or @c wchar_t.
 *
 * @param T is the target type of the conversion
 * @param NAME is the base name for the function
 **/
#define _TG_STR_F_FUNC(T, NAME) \
  _TG_STR_F_FUNC_(T, char, str ## NAME); \
  _TG_STR_F_FUNC_(T, wchar_t, wcs ## NAME)

/* generate the three groups of floating point conversion functions */
```
/**
 ** @brief An internal \texttt{X}macro to generate const conserving string
 ** to integer conversion function.
 **
 ** This macro just assembles \_\texttt{TG\_STR\_I\_FUNC\_}() for the two case
 ** of \texttt{C} being \texttt{char} or \texttt{wchar\_t}.
 **
 ** @param \texttt{T} is the target type of the conversion
 ** @param \texttt{name} is the base name for the function
 **
 ** @define \_\texttt{TG\_STR\_I\_FUNC\_}(T, NAME)
 ** \_\texttt{TG\_STR\_I\_FUNC\_}(T, char, str ## \texttt{name});
 ** \_\texttt{TG\_STR\_I\_FUNC\_}(T, wchar\_t, wcs ## \texttt{name})
 **
 ** generate the six groups of integer conversion functions */

/* generate the six groups of integer conversion functions */
\_\texttt{TG\_STR\_I\_FUNC\_}(long, tol);
\_\texttt{TG\_STR\_I\_FUNC\_}(long long, toll);
\_\texttt{TG\_STR\_I\_FUNC\_}(unsigned long, toul);
\_\texttt{TG\_STR\_I\_FUNC\_}(unsigned long long, toull);
\_\texttt{TG\_STR\_I\_FUNC\_}(intmax\_t, toimax);
\_\texttt{TG\_STR\_I\_FUNC\_}(uintmax\_t, toumax);

/**
 ** @brief An internal macro to select a const conserving string
 ** to integer conversion function.
 **
 ** This macro simply selects the function that corresponds to
 ** the type of argument \texttt{X}. It is not meant to be used
 ** directly but to be integrated in specific macros that
 ** overload \texttt{C} library functions.
 **
 ** The argument \texttt{X} must correspond to an array or pointer,
 ** otherwise an error is triggered in the controlling
There is no default case, because these functions need a string and not some arbitrary bytes. @see strtol on how to integrate this into a type generic user interface.

# define _TG_STR_FUNC3(NAME, X, P, B, ...)  
_Generic(&(*P)[0]),  
  char*: _TG_str ## NAME ## _nc,  
  char const*: _TG_str ## NAME ## _c,  
 wchar_t*: _TG_wcs ## NAME ## _nc,  
 wchar_t const*: _TG_wcs ## NAME ## _c  
)( (X), (P), (B) )

/**  
** @brief An internal macro to select a const conserving string to floating point conversion function. @see _TG_STR_FUNC3 for a complete description of the strategy.  
**
# define _TG_STR_FUNC2(NAME, X, P, ...)  
_Generic(&(*P)[0]),  
  char*: _TG_str ## NAME ## _nc,  
  char const*: _TG_str ## NAME ## _c,  
 wchar_t*: _TG_wcs ## NAME ## _nc,  
 wchar_t const*: _TG_wcs ## NAME ## _c  
)( (X), (P) )

# undef strtod  
# define strtod(...) _TG_STR_FUNC2(tod, __VA_ARGS__, 0, 0, 0)

# undef strtof  
# define strtof(...) _TG_STR_FUNC2(tof, __VA_ARGS__, 0, 0, 0)

# undef strtold  
# define strtold(...) _TG_STR_FUNC2(told, __VA_ARGS__, 0, 0, 0)

# undef strtoul  
# define strtoul(...) _TG_STR_FUNC3(toul, __VA_ARGS__, 0, 0, 0)

# undef strtoimax  
# define strtoimax(...) _TG_STR_FUNC3(toimax, __VA_ARGS__, 0, 0, 0)

# undef strtokmax  
# define strtokmax(...) _TG_STR_FUNC3(tokmax, __VA_ARGS__, 0, 0, 0)