WIKIPEDIA

# Cyclone (programming language)

The **Cyclone** programming language is intended to be a safe dialect of the C language. Cyclone is designed to avoid buffer overflows and other vulnerabilities that are possible in C programs, without losing the power and convenience of C as a tool for system programming.

Cyclone development was started as a joint project of AT&T Labs Research and Greg Morrisett's group at Cornell in 2001. Version 1.0 was released on May 8, 2006.

**Cyclone**

| | |
|---|---|
| **Designed by** | AT&T Labs |
| **First appeared** | 2002 |
| **Final release** | 1.0 / May 8, 2006 |
| **Website** | cyclone.thelanguage.org (http://cyclone.thelanguage.org) |
| **Influenced by** | |
| C | |
| **Influenced** | |
| Rust, Project Verona | |

## Contents

## Language features

Cyclone attempts to avoid some of the common pitfalls of C, while still maintaining its look and performance. To this end, Cyclone places the following limits on programs:

- `NULL` checks are inserted to prevent segmentation faults
- Pointer arithmetic is limited
- Pointers must be initialized before use (this is enforced by definite assignment analysis)
- Dangling pointers are prevented through region analysis and limits on `free()`
- Only "safe" casts and unions are allowed
- `goto` into scopes is disallowed
- `switch` labels in different scopes are disallowed
- Pointer-returning functions must execute `return`
- `setjmp` and `longjmp` are not supported

To maintain the tool set that C programmers are used to, Cyclone provides the following extensions:

- Never-`NULL` pointers do not require `NULL` checks
- "Fat" pointers support pointer arithmetic with run-time bounds checking
- Growable regions support a form of safe manual memory management
- Garbage collection for heap-allocated values
- Tagged unions support type-varying arguments

- Injections help automate the use of tagged unions for programmers
- Polymorphism replaces some uses of `void *`
- varargs are implemented as fat pointers
- Exceptions replace some uses of `setjmp` and `longjmp`

For a better high-level introduction to Cyclone, the reasoning behind Cyclone and the source of these lists, see this paper (http://www.cs.umd.edu/projects/cyclone/papers/cyclone-safety.pdf).

Cyclone looks, in general, much like C, but it should be viewed as a C-like language.

## Pointer types

Cyclone implements three kinds of pointer:

- `*` (the normal type)
- `@` (the never-`NULL` pointer), and
- `?` (the only type with pointer arithmetic allowed, "fat" pointers).

The purpose of introducing these new pointer types is to avoid common problems when using pointers. Take for instance a function, called `foo` that takes a pointer to an int:

```
int foo(int *);
```

Although the person who wrote the function `foo` could have inserted `NULL` checks, let us assume that for performance reasons they did not. Calling `foo(NULL);` will result in undefined behavior (typically, although not necessarily, a SIGSEGV signal being sent to the application). To avoid such problems, Cyclone introduces the `@` pointer type, which can never be `NULL`. Thus, the "safe" version of `foo` would be:

```
int foo(int @);
```

This tells the Cyclone compiler that the argument to `foo` should never be `NULL`, avoiding the aforementioned undefined behavior. The simple change of `*` to `@` saves the programmer from having to write `NULL` checks and the operating system from having to trap `NULL` pointer dereferences. This extra limit, however, can be a rather large stumbling block for most C programmers, who are used to being able to manipulate their pointers directly with arithmetic. Although this is desirable, it can lead to buffer overflows and other "off-by-one"-style mistakes. To avoid this, the `?` pointer type is delimited by a known bound, the size of the array. Although this adds overhead due to the extra information stored about the pointer, it improves safety and security. Take for instance a simple (and naïve) `strlen` function, written in C:

```
int strlen(const char *s)
{
    int iter = 0;
    if (s == NULL)
       return 0;
    while (s[iter] != '\0') {
       iter++;
    }
    return iter;
}
```

This function assumes that the string being passed in is terminated by NULL (`'\0'`). However, what would happen if `char buf[6] = {'h','e','l','l','o','!'};` were passed to this string? This is perfectly legal in C, yet would cause `strlen` to iterate through memory not necessarily associated with the string `s`. There are functions, such as `strnlen` which can be used to avoid such problems, but these functions are not standard with every implementation of ANSI C. The Cyclone version of `strlen` is not so different from the C version:

```
int strlen(const char ? s)
{
   int iter, n = s.size;
   if (s == NULL)
      return 0;
   for (iter = 0; iter < n; iter++, s++) {
      if (*s == '\0')
         return iter;
   }
   return n;
}
```

Here, `strlen` bounds itself by the length of the array passed to it, thus not going over the actual length. Each of the kinds of pointer type can be safely cast to each of the others, and arrays and strings are automatically cast to `?` by the compiler. (Casting from `?` to `*` invokes a bounds check, and casting from `?` to `@` invokes both a `NULL` check and a bounds check. Casting from `*` to `?` results in no checks whatsoever; the resulting `?` pointer has a size of 1.)

## Dangling pointers and region analysis

Consider the following code, in C:

```
char *itoa(int i)
{
   char buf[20];
   sprintf(buf,"%d",i);
   return buf;
}
```

Function `itoa` allocates an array of chars `buf` on the stack and returns a pointer to the start of `buf`. However the memory used on the stack for `buf` is deallocated when the function returns, so the returned value cannot be used safely outside of the function. While gcc and other compilers will warn about such code, the following will typically compile without warnings:

```
char *itoa(int i)
{
   char buf[20], *z;
   sprintf(buf,"%d",i);
   z = buf;
   return z;
}
```

gcc can produce warnings for such code as a side-effect of option -O2 or -O3, but there are no guarantees that all such errors will be detected. Cyclone does regional analysis of each segment of code, preventing dangling pointers, such as the one returned from this version of `itoa`. All of the local variables in a given scope are considered to be part of the same region, separate from the heap or any other local region. Thus, when analyzing `itoa`, the Cyclone compiler would see that `z` is a pointer into the local stack, and would report an error.

## See also

- C
- ML
- Rust

## References

- Cyclone User Manual (http://cyclone.thelanguage.org/wiki/User%20Manual)
- Cyclone: a Type-safe Dialect of C (http://www.cs.umd.edu/~mwh/papers/cyclone-cuj.pdf) by Dan Grossman, Michael Hicks, Trevor Jim, and Greg Morrisett - published January 2005

## External links

- Cyclone Homepage (http://cyclone.thelanguage.org/)
- Old web site (https://web.archive.org/web/20111227232825/http://www.eecs.harvard.edu/~greg/cyclone/old_cyclone.html) since official web site is not available.
- Cyclone - Source code repositories (http://cyclone.thelanguage.org/wiki/Download)
- Cyclone - FAQ (http://cyclone.thelanguage.org/wiki/Frequently%20Asked%20Questions)
- Cyclone for C programmers (http://cyclone.thelanguage.org/wiki/Cyclone%20for%20C%20Programmers)

Presentations:

- Cyclone: A Type-Safe Dialect of C (https://web.archive.org/web/20110607170455/http://www.cs.kent.ac.uk/people/staff/rej/morrisett-4.2.03.ppt)
- Cyclone: A Memory-Safe C-Level Programming Language (http://www.cs.washington.edu/homes/djg/slides/grossman_cyclone_jpl_05.ppt)