What is the difference between char * const and const char *?

Asked 10 years, 5 months ago Active 4 months ago Viewed 180k times

258

149

What's the difference between:

```
char * const
```

and

```
const char *
```

c pointers const

edited Nov 27 '12 at 15:46
amiregelz
1,253 ● 6 ● 18 ● 31

asked May 20 '09 at 22:16
LB.
5,657 ● 20 ● 58 ● 92

1 Possible duplicate of What is the difference between const int*, const int * const, and int const *? – emlai Feb 14 '16 at 13:07

6 The first thing to the left of the "const" is what's constant. If "const" is the thing the farthest to the left, then the first thing to the right of it is what's constant. – Ngineer Jul 31 '16 at 4:43

4 As a friendly tip, never forget that cdecl is a thing. – Braden Best Jun 16 '18 at 7:42

There is another char const* which is the return type of exception::what() – Zhang Jul 22 at 3:14

18 Answers

336

✓

The difference is that `const char *` is a pointer to a `const char`, while `char * const` is a constant pointer to a `char`.

The first, the value being pointed to can't be changed but the pointer can be. The second, the value being pointed at can change but the pointer can't (similar to a reference).

There is also a

```
const char * const
```

which is a constant pointer to a constant char (so nothing about it can be changed).

Note:

The following two forms are equivalent:

```
const char *
```

and

```
char const *
```

The exact reason for this is described in the C++ standard, but it's important to note and avoid the confusion. I know several coding standards that prefer:

```
char const
```

over

```
const char
```

(with or without pointer) so that the placement of the `const` element is the same as with a pointer `const`.

edited Feb 28 '14 at 12:02

answered May 20 '09 at 22:21
workmad3
21.6k ●3 ●32 ●54

6 Would it be worthwhile to note what happens if multiple variables are specified in the same declaration? I believe `const int *foo,*bar;` would declare both `foo` and `bar` to be `int const *`, but `int const *foo, *bar` would declare `foo` to be a `int const *` and `bar` to be `int *`. I think `typedef int * intptr; const intptr foo,bar;` would declare both variables to be `int * const`; I don't know any way to use a combined declaration to create two variables of that type without a typedef. – supercat Apr 12 '13 at 21:57

1 @supercat `I believe const int *foo,*bar; would declare both foo and bar to be int const *`: Yes. `but int const *foo, *bar would declare foo to be a int const * and bar to be int *`: *No!* It would be exactly the same as the previous case. (See ideone.com/RsaB7n where you get the same error for both foo and bar). `I think typedef int * intptr; const intptr foo,bar; would declare both variables to be int * const`: Yes. `I don't know any way to use a combined declaration to create two variables of that type without a typedef`: Well, `int *const foo, *const bar;`. C declarator syntax... – gx_ Aug 28 '13 at 18:35

@gx_: So I was wrong--my uncertainty was why I suggested that it might be helpful to say what the rules are. What would `int const *foo, *volatile bar` do to `bar`? Make it both `const` and `volatile`? I miss Pascal's clean separation of declared-variable names and their types (a pointer to an array of pointers to integers would be `var foo: ^Array[3..4] of ^Integer`;`. That'd be some funny nested parenthesized thing in C, I think. – supercat Aug 28 '13 at 18:54

3 @supercat (oh, C-only, sorry for the C++ code link, I got here from a C++ question) It's all about the *C declaration syntax*, with a ("pure") *type* part followed by a *declarator*. In "`int const *foo, *volatile bar`" the type part is `int const` (stops before the `*`) and the declarators are `*foo` (the expression `*foo` will denote an `int const`) and `*volatile bar`; reading *right-to-left* (good rule for *cv-qualifiers*), `foo` is a pointer to a *const* int, and `bar` is a *volatile* pointer to a *const* int (the pointer itself is volatile, the pointed int is [accessed as] const). – gx_ Aug 28 '13 at 21:23

@supercat And as for "a pointer to an array of pointers to integers" (I don't know Pascal, not sure about the `[3..4]` syntax, so let's take an array of 10 elements): `int *(*foo)[10];`. It mirrors its (future) use as an expression: `*(*foo)[i]` (with `i` an integer in the range `[0, 10)` i.e. `[0, 9]`) will first dereference `foo` to get at the array, then access the element at index `i` (because postfix `[]` binds tighter than prefix `*`), then dereference this element, finally yielding an `int` (see ideone.com/jgjIjR). But `typedef` makes it easier (see ideone.com/O3wb7d). – gx_ Aug 28 '13 at 21:25

94

To avoid confusion, always *append* the const qualifier.

```
int       *      mutable_pointer_to_mutable_int;
int const *      mutable_pointer_to_constant_int;
int       *const constant_pointer_to_mutable_int;
int const *const constant_pointer_to_constant_int;
```

answered May 21 '09 at 0:08
diapir
1,974 ●12 ●22

9 Why? "To avoid confusion" doesn't explain what the confusion is to me. – Andrew Weir Nov 20 '13 at 11:48

11 @Andrew: I was hinting at consistency and thus readability. Writing all type qualifiers so they modify what's on their left, *always*, is what I use. – diapir Nov 20 '13 at 14:31

1 Gotcha. Thanks. – Andrew Weir Nov 20 '13 at 16:51

8 As a code standard, I have rarely encountered this style and so am not likely to adopt it. However as a learning tool, this answer was very helpful! (So I guess too bad this isn't more common style.) – natevw Sep 29 '14 at 21:33

7 @Alla: `p` doesn't relate to the type : `(const int *const)`. For better or worse (worse if you ask me) the const qualifier, both in C and C++, is meant to be postfix : cf const member function `void foo(int a) const;`. The possibility to declare `const int` is the exception rather than the rule. – diapir Apr 9 '15 at 7:17

41

`const` always modifies the thing that comes before it (to the left of it), EXCEPT when it's the first thing in a type declaration, where it modifies the thing that comes after it (to the right of it).

So these two are the same:

```
int const *i1;
const int *i2;
```

they define pointers to a `const int`. You can change where `i1` and `i2` points, but you can't change the value they point at.

This:

```
int *const i3 = (int*) 0x12345678;
```

defines a `const` pointer to an integer and initializes it to point at memory location 12345678. You can change the `int` value at address 12345678, but you can't change the address that `i3` points to.

answered May 20 '09 at 22:36
Don McCaughey
7,577 ● 3 ● 25 ● 35

22

`const * char` is invalid C code and is meaningless. Perhaps you meant to ask the difference between a `const char *` and a `char const *`, or possibly the difference between a `const char *` and a `char * const`?

See also:

- What are const pointers (as opposed to pointers to const objects)?
- Const in C
- Difference between const declarations in C++
- C++ const question
- Why can I change the values of a const char* variable?

edited May 23 '17 at 12:03
Community ♦
1 ● 1

answered May 20 '09 at 22:22
Adam Rosenfield
322k ● 85 ● 459 ● 554

14

`const char*` is a pointer to a constant character
`char* const` is a constant pointer to a character
`const char* const` is a constant pointer to a constant character

edited Jan 13 '12 at 15:19

answered May 20 '09 at 22:20
Andrew Coleson
5,944 ●5 ●26 ●30

9

Rule of thumb: read the definition from right to left!

`const int *foo;`

Means " `foo` points (`*`) to an `int` that cannot change (`const`)".
To the programmer this means "I will not change the *value* of what `foo` points to".

- `*foo = 123;` or `foo[0] = 123;` would be invalid.
- `foo = &bar;` is allowed.

`int *const foo;`

Means " `foo` cannot change (`const`) and points (`*`) to an `int` ".
To the programmer this means "I will not change the *memory address* that `foo` refers to".

- `*foo = 123;` or `foo[0] = 123;` is allowed.
- `foo = &bar;` would be invalid.

`const int *const foo;`

Means " `foo` cannot change (`const`) and points (`*`) to an `int` that cannot change (`const`)".
To the programmer this means "I will not change the *value* of what `foo` points to, nor will I change the *address* that `foo` refers to".

- `*foo = 123;` or `foo[0] = 123;` would be invalid.
- `foo = &bar;` would be invalid.

answered Jul 8 '16 at 0:59
Mr. Llama
16.2k ●2 ●43 ●83

8

1. **const char* x** Here X is basically a character pointer which is pointing to a constant value
2. **char* const x** is refer to character pointer which is constant, but the location it is pointing can be change.
3. **const char* const x** is combination to 1 and 2, means it is a constant character pointer which is pointing to constant value.
4. **const *char x** will cause a compiler error. it can not be declared.
5. **char const * x** is equal to point 1.

the rule of thumb is if **const** is with var name then the *pointer will be constant but the pointing location can be changed* , else *pointer will point to a constant location and pointer can point to another location but the pointing* *location content can not be change*.

edited Jun 5 at 22:49
Evan Carroll
39.8k ● 27 ● 157 ● 269

answered Jul 4 '12 at 18:19
AAnkit
23.3k ● 10 ● 51 ● 63

1 "char* const x is refer to character pointer which is constant, but the location it is pointing can be change." Wrong. The value at the location can be changed not the location itself. – PleaseHelp Mar 12 '15 at 13:44

3

Another thumb rule is to check where **const is**:

1. **before** * => **value** stored is **constant**
2. **after** * => **pointer** itself is **constant**

answered Jan 25 '13 at 9:43
Aadishri
745 ● 2 ● 11 ● 26

3

First one is a syntax error. Maybe you meant the difference between

```
const char * mychar
```

and

```
char * const mychar
```

In that case, the first one is a pointer to data that can't change, and the second one is a pointer that will always point to the same address.

answered May 20 '09 at 22:21
Javier
2,798 ● 5 ● 27 ● 44

2

Lots of answer provide specific techniques, rule of thumbs etc to understand this particular instance of variable declaration. But there is a generic technique of understand any declaration:

> ## Clockwise/Spiral Rule

A)

```
const char *a;
```

As per the clockwise/spiral rule `a` is pointer to character that is constant. Which means character is constant but the pointer can change. i.e. `a = "other string";` is fine but `a[2] = 'c';` will fail to compile

B)

```
char * const a;
```

As per the rule, `a` is const pointer to a character. i.e. You can do `a[2] = 'c';` but you cannot do `a = "other string";`

answered Apr 14 '17 at 18:01
PnotNP
1,910 ● 1 ● 18 ● 37

1 Also known as right-left rule (at least that's how I learnt it): jdurrett.ba.ttu.edu/3345/handouts/RL-rule.html – pruzinat May 9 '18 at 12:05

(Would be much better if the essence of the answer would not be hidden behind a link, with the text here not even citing, or at least referring, to any of its specifics, beyond a generic "as per the rule".) – Sz. Jul 14 at 9:58

@Sz. Do you have any specific confusion here that I can clear? There is really not much to it after knowing the rule. – PnotNP Jul 15 at 1:53

1

I presume you mean const char * and char * const .

The first, const char *, is a pointer to a constant character. The pointer itself is mutable.

The second, char * const is a constant pointer to a character. The pointer cannot change, the character it points to can.

And then there is const char * const where the pointer and character cannot change.

answered May 20 '09 at 22:21
Michael
48.3k ● 5 ● 103 ● 133

Your first two are actually the same and your third is a compiler error :) – workmad3 May 20 '09 at 22:22

Whoops, you are right. Fixed. – Michael May 20 '09 at 22:23

1

char * const and const char *?

1. Pointing to a constant value

`const char * p;` // value cannot be changed

2. Constant pointer to a value

`char * const p;` // address cannot be changed

3. Constant pointer to a constant value

`const char * const p;` // both cannot be changed.

edited Nov 27 '15 at 11:03

answered Nov 27 '15 at 10:49
Yogeesh H T
1,539 ● 16 ● 17

1

```
// Some more complex constant variable/pointer declaration.
// Observing cases when we get error and warning would help
// understanding it better.

int main(void)
{
  char ca1[10]= "aaaa"; // char array 1
  char ca2[10]= "bbbb"; // char array 2

  char *pca1= ca1;
  char *pca2= ca2;

  char const *ccs= pca1;
  char * const csc= pca2;
  ccs[1]='m';  // Bad - error: assignment of read-only location ‘*(ccs + 1u)’
  ccs= csc;    // Good

  csc[1]='n';  // Good
  csc= ccs;    // Bad - error: assignment of read-only variable ‘csc’

  char const **ccss= &ccs;     // Good
  char const **ccss1= &csc;    // Bad - warning: initialization from
incompatible pointer type

  char * const *cscs= &csc;    // Good
  char * const *cscs1= &ccs;   // Bad - warning: initialization from
incompatible pointer type

  char ** const cssc=   &pca1; // Good
  char ** const cssc1=  &ccs;  // Bad - warning: initialization from
incompatible pointer type
  char ** const cssc2=  &csc;  // Bad - warning: initialization discards ‘const’
                               //                qualifier from pointer target
type

  *ccss[1]= 'x'; // Bad - error: assignment of read-only location ‘**(ccss +
8u)’
  *ccss= ccs;    // Good
  *ccss= csc;    // Good
  ccss= ccss1;   // Good
  ccss= cscs;    // Bad - warning: assignment from incompatible pointer type

  *cscs[1]= 'y'; // Good
  *cscs= ccs;    // Bad - error: assignment of read-only location ‘*cscs’
  *cscs= csc;    // Bad - error: assignment of read-only location ‘*cscs’
  cscs= cscs1;   // Good
  cscs= cssc;    // Good

  *cssc[1]= 'z'; // Good
  *cssc= ccs;    // Bad - warning: assignment discards ‘const’
                 //                qualifier from pointer target type
  *cssc= csc;    // Good
  *cssc= pca2;   // Good
  cssc= ccss;    // Bad - error: assignment of read-only variable ‘cssc’
  cssc= cscs;    // Bad - error: assignment of read-only variable ‘cssc’
  cssc= cssc1;   // Bad - error: assignment of read-only variable ‘cssc’
}
```

answered Apr 23 '15 at 17:38

gopalshankar
11 ●2

1

I would like to point out that using `int const *` (or `const int *`) isn't about a pointer pointing to a `const int` variable, but that this variable is `const` for this specific pointer.

For example:

```
int var = 10;
int const * _p = &var;
```

The code above compiles perfectly fine. `_p` points to a `const` variable, although `var` itself isn't constant.

edited Jan 20 '18 at 23:00

answered Jan 20 '18 at 15:49
SteliosKts
23 ●1 ●5

1

Here is a detailed explanation with code

```
/*const char * p;
char * const p;
const char * const p;*/ // these are the three conditions,

// const char *p;const char * const p; pointer value cannot be changed

// char * const p; pointer address cannot be changed

// const char * const p; both cannot be changed.

#include<stdio.h>

/*int main()
{
    const char * p; // value cannot be changed
    char z;
    //*p = 'c'; // this will not work
    p = &z;
    printf(" %c\n",*p);
    return 0;
}*/

/*int main()
{
    char * const p; // address cannot be changed
    char z;
    *p = 'c';
    //p = &z;   // this will not work
    printf(" %c\n",*p);
    return 0;
}*/



/*int main()
{
    const char * const p; // both address and value cannot be changed
    char z;
    *p = 'c'; // this will not work
    p = &z; // this will not work
    printf(" %c\n",*p);
    return 0;
}*/
```

edited Apr 12 '13 at 21:49
Reese Moore
10.4k ●3 ●18 ●29

answered Mar 4 '13 at 10:21
Megharaj
955 ●16 ●31

@reese moore Thank you. – Megharaj Apr 18 '13 at 6:48

1

The `const` modifier is applied to the term immediately to its left. The only exception to this is when there is nothing to its left, then it applies to what is immediately on its right.

These are all equivalent ways of saying "constant pointer to a constant `char`":

- `const char * const`
- `const char const *`
- `char const * const`
- `char const const *`

answered Jul 21 '16 at 4:31
galois
490 ●1 ●5 ●22

Is it compiler dependent? gcc produce for "const char const *" and "const const char *" and "char const const *" the same result -> pointer could pointing to other location. – cosinus0 Aug 4 '17 at 8:57

1

1. **Constant pointer**: A constant pointer can point only to a single variable of the respective data type during the entire program.we can change the value of the variable pointed by the pointer. Initialization should be done during the time of declaration itself.

Syntax:

```
datatype *const var;
```

`char *const` comes under this case.

```
/*program to illustrate the behaviour of constant pointer */

#include<stdio.h>
int main(){
  int a=10;
  int *const ptr=&a;
  *ptr=100;/* we can change the value of object but we cannot point it to
another variable.suppose another variable int b=20; and ptr=&b; gives you
error*/
  printf("%d",*ptr);
  return 0;
}
```

2. **Pointer to a const value**: In this a pointer can point any number of variables of the respective type but we cannot change the value of the object pointed by the pointer at that specific time.

Syntax:

`const datatype *var` or `datatype const *var`

`const char*` comes under this case.

```
/* program to illustrate the behavior of pointer to a constant*/

   #include<stdio.h>
   int main(){
       int a=10,b=20;
       int const *ptr=&a;
       printf("%d\n",*ptr);
       /*  *ptr=100 is not possible i.e we cannot change the value of the object
pointed by the pointer*/
       ptr=&b;
       printf("%d",*ptr);
       /*we can point it to another object*/
       return 0;
    }
```

edited Oct 23 '15 at 14:23
Ram
2,679 ● 9 ● 33 ● 55

answered Oct 23 '15 at 13:55
Goutham Gundapu
11 ● 1

1

Two rules

1. `If const is between char and *, it will affect the left one.`
2. `If const is not between char and *, it will affect the nearest one.`

e.g.

1. `char const *. This is a pointer points to a constant char.`
2. `char * const. This is a constant pointer points to a char.`

edited Nov 17 '16 at 7:11
Jishnu V S
6,564 ● 6 ● 19 ● 48

answered Nov 17 '16 at 6:14
Xinpei Zhai
11 ● 2

protected by Sheldore Jul 19 at 10:44

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?