Meaning of *& and **& in C++

Asked 11 years ago Modified 2 months ago Viewed 128k times

84

I found these symbols in a function declaration several times, but I don't know what they mean.

Example:

```
void raccogli_dati(double **& V, double **p, int N) {
  int ultimo = 3;
  V = new double * [N/2];
  for(int i=0; i < N/2; i++) {
     V[i] = new double[N/2], std :: clog << "digita " << N/2 - i
                 << " valori per la parte superiore della matrice V: ";
    for(int j=i; j < N/2; j++)
      std :: cin >> V[i][j], p[ultimo++][0] = (V[i][j] /= sqrt(p[i][0]*p[j][0]));
  }
  for(int i=1; i < N/2; i++)
    for(int j=0; j < i; j++)
      V[i][j] = V[j][i];
}
```

c++ pointers syntax reference symbols

Share Edit Follow Flag

edited Aug 17, 2016 at 5:28 fragiewindows 1,354 ●1 ●14 ●26

asked Apr 26, 2011 at 11:46 sdffadsf 1,002 ●2 ●9 ●8

In actual code or a book? To me it just looks like notation for a general function. One which returns a single pointer, and the other a pointer to a pointer. – Mr. Shickadance Apr 26, 2011 at 11:48

Those are references to pointers. – Alexandre C. Apr 26, 2011 at 11:49

2 This website may be helpful to you.) – JW.ZG Feb 11, 2016 at 0:39

8 Answers

Sorted by: Highest score (default)

127

That is taking the parameter by reference. So in the first case you are taking a pointer parameter by reference so whatever modification you do to the value of the pointer is reflected outside the function. Second is the similiar to first one with the only difference being that it is a double pointer. See this example:

```
void pass_by_value(int* p)
{
    //Allocate memory for int and store the address in p
    p = new int;
}

void pass_by_reference(int*& p)
{
    p = new int;
}

int main()
{
    int* p1 = NULL;
    int* p2 = NULL;

    pass_by_value(p1); //p1 will still be NULL after this call
    pass_by_reference(p2); //p2 's value is changed to point to the newly allocate
memory

    return 0;
}
```

Share Edit Follow Flag

answered Apr 26, 2011 at 11:49 Naveen 71.7k ●44 ●171 ●229

24

First is a reference to a pointer, second is a reference to a pointer to a pointer. See also FAQ on how pointers and references differ.

```
void foo(int*& x, int**& y) {
  // modifying x or y here will modify a or b in main
}

int main() {
    int val = 42;
    int *a  = &val;
    int **b = &a;

    foo(a, b);
    return 0;
}
```

Share Edit Follow Flag

edited May 23, 2017 at 10:30 Community Bot 1 ●1

answered Apr 26, 2011 at 11:50 Cat Plus Plus 120k ●26 ●191 ●218

4 I know I shouldn't really, but I upvoted this answer simply because it contains an a-pointer-a-pointer-to-a-reference-to-an-allusion-to-the-meaning-of-life. Somehow I think we're a few years away from Deep Thought though. – corlettk Apr 26, 2011 at 12:18

17

That's passing a pointer by reference rather than by value. This for example allows altering the pointer (not the pointed-to object) in the function is such way that the calling code sees the change.

Compare:

```
void nochange( int* pointer ) //passed by value
{
   pointer++; // change will be discarded once function returns
}

void change( int*& pointer ) //passed by reference
{
   pointer++; // change will persist when function returns
}
```

Share Edit Follow Flag

answered Apr 26, 2011 at 11:49 sharptooth 163k ●92 ●499 ●939

9

An `int*` is a pointer to an `int`, so `int*&` must be a reference to a pointer to an `int`. Similarly, `int**` is a pointer to a pointer to an `int`, so `int**&` must be a reference to a pointer to a pointer to an `int`.

Share Edit Follow Flag

edited Apr 5, 2019 at 22:05 S.S. Anne 14.4k ●7 ●35 ●68

answered Apr 26, 2011 at 11:50 Oswald 30.4k ●3 ●40 ●68

1 i understand their literal meanings, can u write an example to illustrate more effectively among them? – Sheldon Jun 5, 2017 at 14:19

6

`*&` signifies the receiving the pointer by reference. It means it is an alias for the passing parameter. So, it affects the passing parameter.

```
#include <iostream>
using namespace std;

void foo(int *ptr)
{
    ptr = new int[50];    // Modifying the pointer to point to a different location
    cout << "In foo:\t" << *ptr << "\n";
    delete ptr ;
}

void bar(int *& ptr)
{
    ptr = new int[80];    // Modifying the pointer to point to a different location
    cout << "In bar:\t" << *ptr << "\n";
    // Deleting the pointer will result the actual passed parameter dangling
}
int main()
{
    int temp = 100 ;
    int *p = &temp ;

    cout << "Before foo:\t" << *p << "\n";
    foo(p) ;
    cout << "After foo:\t" << *p << "\n";

    cout << "Before bar:\t" << *p << "\n";
    bar(p) ;
    cout << "After bar:\t" << *p << "\n";

    delete p;

    return 0;
}
```

Output:

```
Before foo: 100
In foo: 50
After foo:  100
Before bar: 100
In bar: 80
After bar:  80
```

Share Edit Follow Flag

answered Apr 26, 2011 at 12:04 Mahesh 33.6k ●17 ●84 ●113

4

To understand those phrases let's look at the couple of things:

```
typedef double Foo;
void fooFunc(Foo &_bar){ ... }
```

So that's passing a double by reference.

```
typedef double* Foo;
void fooFunc(Foo &_bar){ ... }
```

now it's passing a pointer to a double by reference.

```
typedef double** Foo;
void fooFunc(Foo &_bar){ ... }
```

Finally, it's passing a pointer to a pointer to a double by reference. If you think in terms of typedefs like this you'll understand the proper ordering of the & and * plus what it means.

Share Edit Follow Flag

answered Apr 26, 2011 at 11:52 wheaties 35k ●10 ●86 ●129

3

Typically, you can read the declaration of the variable from right to left. Therefore in the case of `int *ptr;` , it means that you have a *Pointer* `*` to an *Integer variable* `int`. Also when it's declared `int **ptr2;`, it is a *Pointer variable* `*` to a *Pointer variable* `*` pointing to an *Integer variable* `int` , which is the same as `*(int *)* ptr2;`

Now, following the syntax by declaring `int*& rPtr;`, we say it's a *Reference* `&` to a *Pointer* `*` that points to a *variable* of type `int`. Finally, you can apply again this approach also for `int**& rPtr2;` concluding that it signifies a *Reference* `&` to a *Pointer* `*` to a *Pointer* `*` to an *Integer* `int`.

Share Edit Follow Flag

answered Jul 24, 2019 at 16:53 Juan Chavarro 33 ●5

2 +1 for the tip to read the declarations from right to left this speeds up understanding greatly, and also explains why interpretations of C++ construct can be difficult sometimes (for those who normally read from left to right at least) – XavierStuvw Feb 20, 2020 at 17:34

2

This ***&** in theory as well as in practical its possible and called as reference to pointer variable. and it's act like same. This ***&** combination is used in as function parameter for 'pass by' type defining. unlike ** can also be used for declaring a double pointer variable.
The passing of parameter is divided into *pass by value, pass by reference, pass by pointer*. there are various answer about "pass by" types available. however the basic we require to understand for this topic is.

pass by reference --> generally operates on already created variable refereed while passing to function e.g `fun(int &a);`

pass by pointer --> Operates on already initialized 'pointer variable/variable address' passing to function e.g `fun(int* a);`

```
auto addControl = [](SomeLabel** label, SomeControl** control) {
    *label = new SomeLabel;
    *control = new SomeControl;
    // few more operation further.
};

addControl(&m_label11,&m_control1);
addControl(&m_label12,&m_control2);
addControl(&m_label13,&m_control3);
```

in the above example(this is the real life problem i came across) i am trying to init few pointer variable from the lambda function and for that we need to pass it by double pointer, so that comes with d-referencing of pointer for its all usage inside of that lambda + while passing pointer in function which takes double pointer, you need to pass reference to the pointer variable.

so with this same thing reference to the pointer variable, ***&** this combination helps. in below given way for the same example i have mentioned above.

```
auto addControl = [](SomeLabel*& label, SomeControl*& control) {
        label = new SomeLabel;
        control = new SomeControl;
        // few more operation further.
    };

addControl(m_label11,m_control1);
addControl(m_label12,m_control2);
addControl(m_label13,m_control3);
```

so here you can see that you neither require d-referencing nor we require to pass reference to pointer variable while passing in function, as current pass by type is already reference to pointer.

Hope this helps :-)

Share Edit Follow Flag

edited Sep 9, 2019 at 9:49

answered Sep 9, 2019 at 4:58 Sachin Nale 119 ●1 ●4