1 . Pointer :

1. **int \*const** q

is a constant pointer to integer .

This means that the variable being declared is a constant pointer pointing to an integer. Effectively, this implies that the pointer shouldn’t point to some other address. Const qualifier doesn’t affect the value of integer in this scenario so the value being stored in the address is allowed to change.

|  |
| --- |
| #include <stdio.h>  int main(){      const int q = 7.5; //Compilation error      int \*const p = &q; //Valid      \*p = 7;      const int q2 = 7; //Compilation error      p = &q2;      return 0;  } |

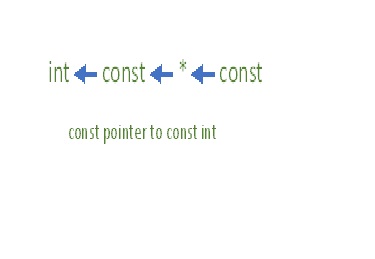
1. **const** **int \* p**

is a constant pointer to constant integer .

This means that the variable being declared is a constant pointer pointing to a constant integer. Effectively, this implies that a constant pointer is pointing to a constant value. Hence, neither the pointer should point to a new address nor the value being pointed to should be changed.  
The first const keyword can go either side of data type, hence **const int\* p** is equivalent to **int const\* p**

|  |  |
| --- | --- |
| #include <stdio.h>  int main(){      const int q = 7.5; //Valid      const int\* const p = &q; //Compilation error      \*p = 7;      const int q2 = 7; //Compilation error      p = &q2;      return 0;  }   1. **int const\* p**   is a constant pointer to constant integer .  This means that the variable being declared is a constant pointer pointing to a constant integer. Effectively, this implies that a constant pointer is pointing to a constant value. Hence, neither the pointer should point to a new address nor the value being pointed to should be changed. The first const keyword can go either side of data type, hence **const int\* p** is equivalent to **int const\* p**   |  | | --- | | #include <stdio.h>  int main(){      const int q = 7.5; //Valid      int\* const p = &q; //Compilation error  \*p = 7;      const int q2 = 7; //Compilation error      p = &q2;      return 0;  } | |

The rule can also be seen as decoding the syntax from right to left.



Hence,

* **int const\*** is **pointer** to **const** **int**
* **int \*const** is **const** **pointer** to **int**
* **int const\* const** is **const** **pointer** to **const** **int**

Using this rule, even complex declarations can be decoded like,

* **int \*\* const** is a **const** **pointer** to **pointer** to an **int**.
* **int \* const \*** is a **pointer** to **const** **pointer** to an **int**.
* **int const \*\*** is a **pointer** to a **pointer** to a **const** **int**.
* **int \* const \* const** is a **const** **pointer** to a **const** **pointer** to an **int**.

2. Pointer as an arguments

Code in C language

#include<stdio.h>

void addOne(int\* ptrNum);

int main(){

int num = 7;

printf("Before: %d\n", num);

addOne(&num);

printf("After: %d\n", num);

return 0;

}

void addOne(int\* ptrNum){

\*ptrNum = \*ptrNum + 1;

}