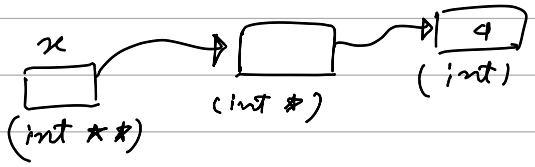
```
int ** x = new int * ;
** x = 4 ;
cout << ***(& x) ⇒ 4
```

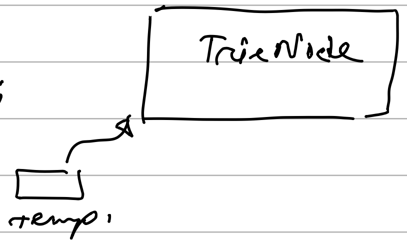
x

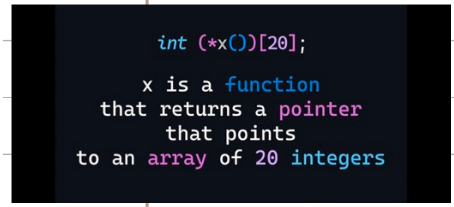
0x100	0x103
0x101	
0x102	4
0x103	0x102
0x104	
0x105	



(&x) = 0x100

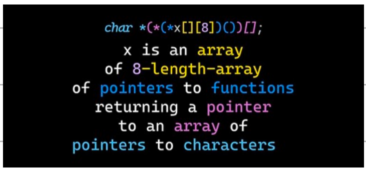
```
TrieNode * temp = new TrieNode ;
```





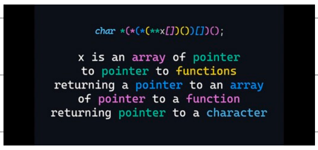
```
int (*x())[20];
```

x is a function
that returns a pointer
that points
to an array of 20 integers



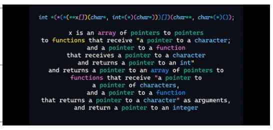
```
char *(*(*x[][8])())[];
```

x is an array
of 8-length-array
of pointers to functions
returning a pointer
to an array of
pointers to characters



```
char *(*(*(**x[])())[])();
```

x is an array of pointer
to pointer to functions
returning a pointer to an array
of pointer to a function
returning pointer to a character



```
int *(*(*(**x[])(char*, int*(*)(char*)))[])(char**, char*(*)());
```

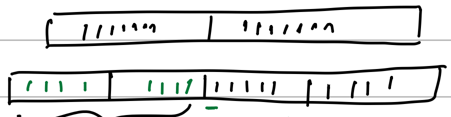
x is an array of pointers to pointers
to functions that receive "a pointer to a character;
and a pointer to a function
that receives a pointer to a character
and returns a pointer to an int"
and returns a pointer to an array of pointers to
functions that receive "a pointer to
a pointer of characters,
and a pointer to a function
that returns a pointer to a character" as arguments,
and return a pointer to an integer

c/c++ :-

bool			
char	1 byte		
short	4 bytes		
int	4 bytes		
long	4 bytes	long long	8 bytes
float	4 bytes		
double	8 bytes		

✓ short s = -1

int i = s

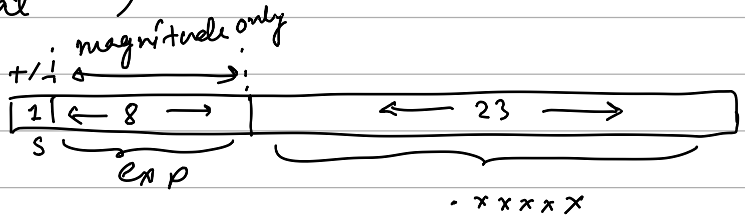


S+1 = 0

filled with all 1's called sign extend

#Term/sign extend

float →



$(-1)^s \; 1.xxxxx * 2^{exp-127}$

7.0

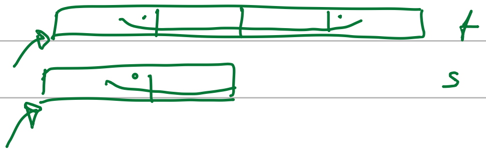
↳ 1.75×2^2 number is made to fit to be able to represent in float format.

# ✓ int i = 5
float f = i } cout << f;
↳ 5 (it converts properly)

but

int i = 37
float f = *(float*) &i } cout << f prints some extremely small number.

✓ float f = 7.0
short s = *(short *) &f



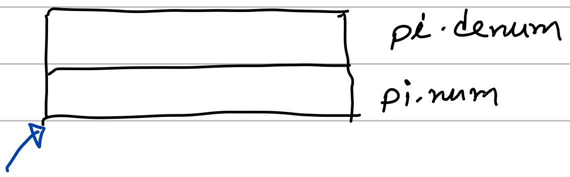
start dereferencing from beginning

#miscellaneous-STL

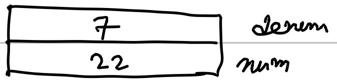
```
struct fraction{
    int num;
    int denum; };
```

#address of the struct is always coincident with the address of the first field

fraction pi;

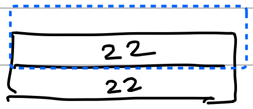


pi.num = 22 ; pi.denum = 7



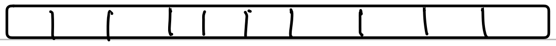
((fraction *) &(pi.denum)) → num = 22

dereferences as a fraction object



---

int array[10]



array is synonymous to the address of the first entry of the array.

array ≡ &array[0]

#Term/pointer arithmetic

array + k ≡ & array[k]

#automatically scaled according to the data type. here it's int.

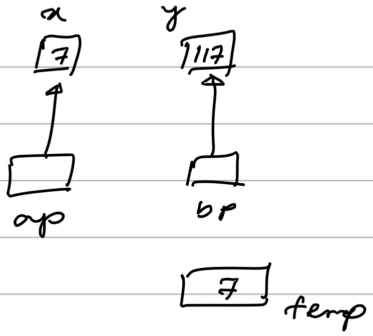
(& pi)[1].num = ... } pi is a fraction object [1] is the next address accordingly.

✓

*array = array[0]

*(array+k) = array[k] scaled accordingly.

```
void  swap( int*  ap,  int* bp)
{       int  temp = *ap;
        *ap  = * bp ;
        *bp  =  temp;
}
swap (&x , &y)
int x =7
int y = 117.
```



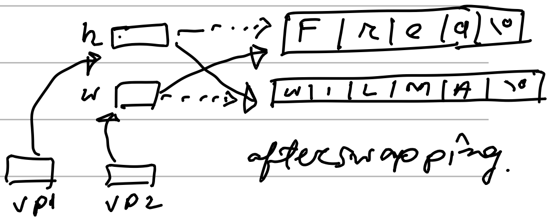
→ writing a generic swap function;

```
void  swap(void * ap , void* bp)
{   void *  temp = * ap;
    *ap  = * bp ;
    *bp  = temp
}
```

this won't work bcz,: → void * type variable is invalid and dereferencing *ap , a void* object is invalid.

```
void  swap(void * vp1 , void* vp2 , int size)
{      char  buffer[size];
       memcpy( buffer, vp1, size);
       memcpy( vp1, vp2, size);
       memcpy( vp2, buffer, size); }
```

Eg h = Fred
    w = Wilma
swap(&h, &w, sizeof(char))



✓ generic linear search.

```
void * lsearch( void * Key , void * base, int n, int elemSize,
                         int (* cmpfn) (void *, void *))
{
  for (int i=0; i<n ; i++)
   { void * elemAddr = (char *) base + i* elemSize ;
     if ( cmpfn( key, elemAddr) == 0 ) return elemAddr;
   }
  return null
}
```