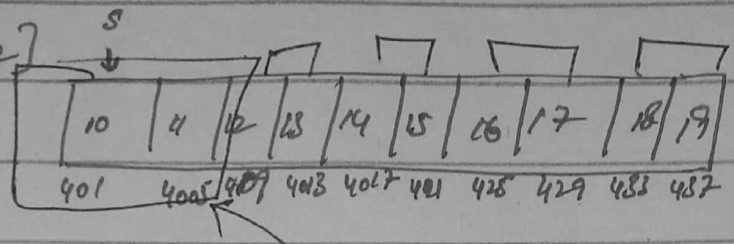
Pointer in C Language

s[5][2]



s[0] = *(s+0) = 401

s[1] = *(s+1) = 409

⇒ By using a[i] we only accessed base addresses i.e; 401, 409 ... 433.

s[4] = *(s+4) = 433

In two dimentional Array

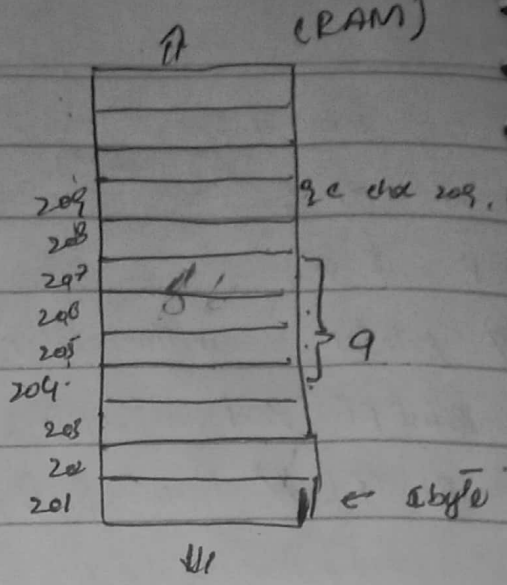
s[2][1] = *(s[2]+1) = *(*(s+2)+1)

int - 4 bytes

char 1 byte

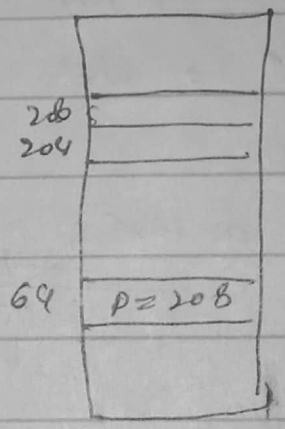
float 4 byte

```
int a;
a = 5;
. . . .
. . . .
a++;
```



Pointer - variable that stores address of another v.

```
int a;
int *p;
p = &a;
Print p.   // 204
Print &a;  // 204
Print &p;  // 64
Print *p;  // 5 ← dereference
*p = 8;    changes
Print a;   // 8
```



```
int  a = 9;
int  *p;
p = &a;
// pointer arithematic
Printf("%d\n", p);  // P is 2002
Printf("%d\n", p+1);  // P+1 = 2006  (TA)
```

کیونکہ Increment، Datatype کے لحاظ سے تبدیل ہوتا ہے

Where

int * → int stored

long int * → long int

char * → char story.

integer

int a = 1025;

byte 4	byte 3	byte 2	byte 0
100000000	000000000	000000000	100000000
203	202	201	200

sign bit

So 1025 = 00000100 | 00000001
(2^{10} … 2^0)

```
int *p;
p = &a;
Printf("%d", P); 200
Print *p;  ⇐
```

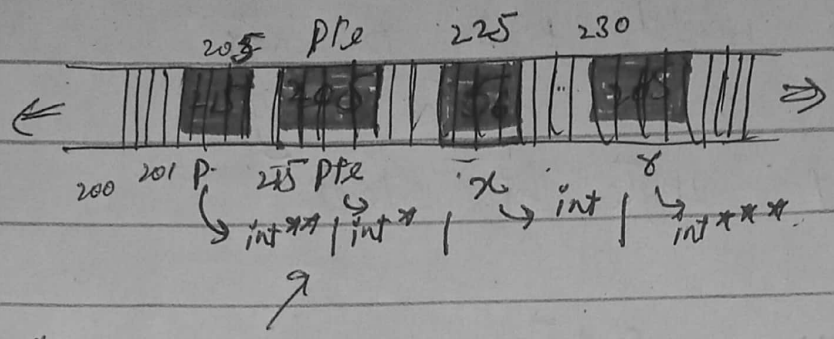
This say that sinse P is point to an integer so look at 4 bytes and extract 1025; which start (200)

If this would pointer to an character then Compiler only see at 1byte that starts 200 or

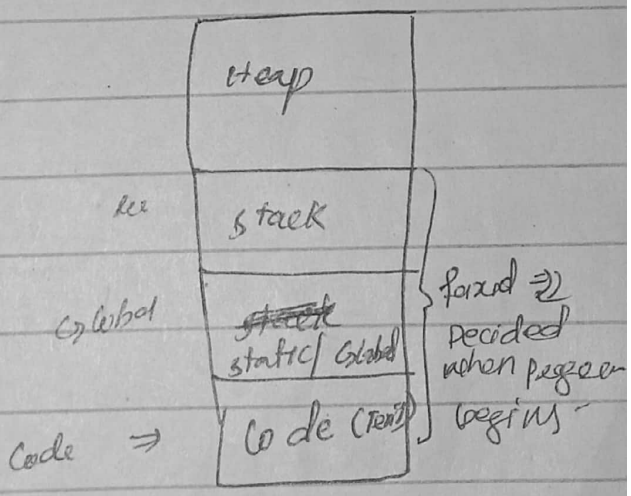
Generic Pointer

```
void *PG;
PG=P;
```

Pointer to Pointer




```
int x=5;
int* ptr;
*p = 6;
int **p;
   p = &ptr;
int *** γ ;
   γ = &q;
```



Pointer and Arrays.

&A[i] = (A+i)

a[i] = *(A+i)

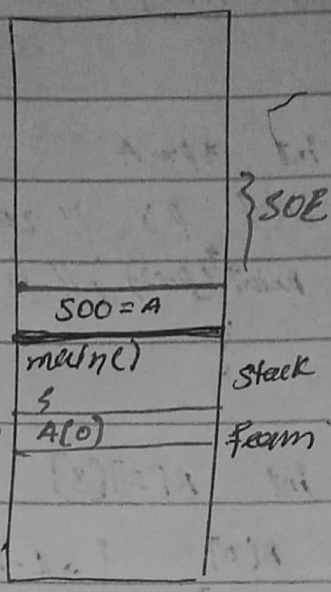
when Array in a function

e.g;

```
int SumOfElements (int A[])
{
}

int main ()
{ A[5] = {1,3,2,4,5};
  SumOfElemnts (A);
}
```

this means in backend- int* A



goes adress

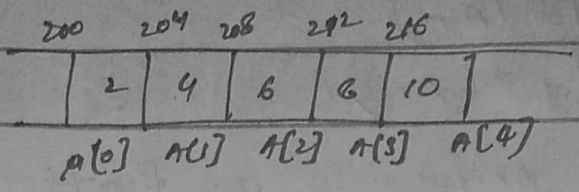
character Arrays and Pointer.

```
void Print (char *C)
{ while (*C != '\0')
  {
    printf("%c", *C);
    C++;
  }
}
```

Print 2

char *P = "Hellow"; // Valid statement

pointers And Multidirectional Array



int A[5];

int *p = A

Print p; // 200

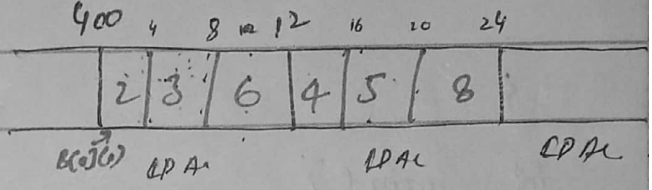
Print (p+2); // 6

⋮

*(A+i) same A[i]

int B[2][3]

B[0] } → 1-D Array of 3 integers
B[1] }



2D-Array

~~int *p = B; // This will point to 1D Array of 8 integers.~~

int *p[3] = B; // valid.

This is pointer Array that point 8 pointers.

Print B or &B[0] // 400

Print *B or B[0] or &B[0][0] // 400

Print B+1; // 412

Print &B[1]; // 412

Print *(B+2); // 412

Print B[1]; // 412

Print &B[1][0]; // 412

B ≅ &B[0]

*B ≅ B[0] or &B[0][0]

(B+1) ≅ &B[1]

*(B+1) or B[1] or &B[1][0]

↳ returning int* to first integer in B[1]

Print *(B+1) +2 ; // 420

↳ int*

↳ B[1]+2 or &B[1][2] //420

*(*B +1);

= *(B[0]+1);

= *(&B[0][1]); = B[0][1] = 3

Note

B[i][j] = *(B[i]+j)

= *(*(B+i)+j)

DYNAMIC MEMORY ALLOCATION

heap = free store of memory (large free pool of memory)

↳ Dynamics memory

* 4- Functions

** malloc → void* malloc(size_t size); (unsigned int *n)

** calloc → void* calloc(nofblock, sizeof(int));

** realloc → void* realloc(void* ptr, nofblock);

** free(ptr);

↳ If free all block of memory and their remain garbage values.

Pointers as Function returns

Let a function

```
int add(int a, int b)
{  return a+b;
}
```

then pointer to this function is

```
int (*p)(int, int)
   p = &add;
```

Then we can call

```
printf("%d", *p(5,6));
```

Function Pointers and Callbacks

Function pointers

↳ can be passed as argument to functions.