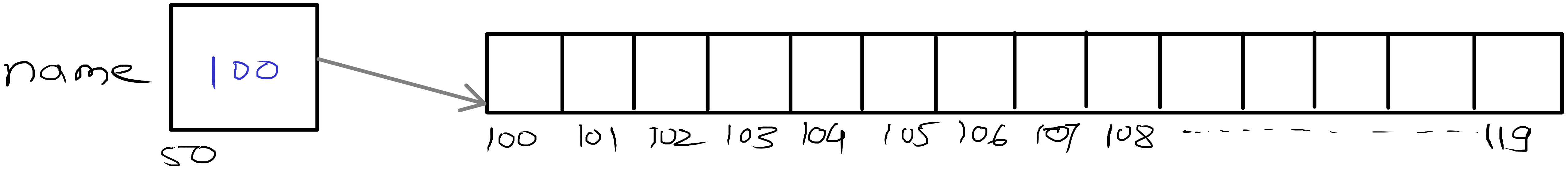
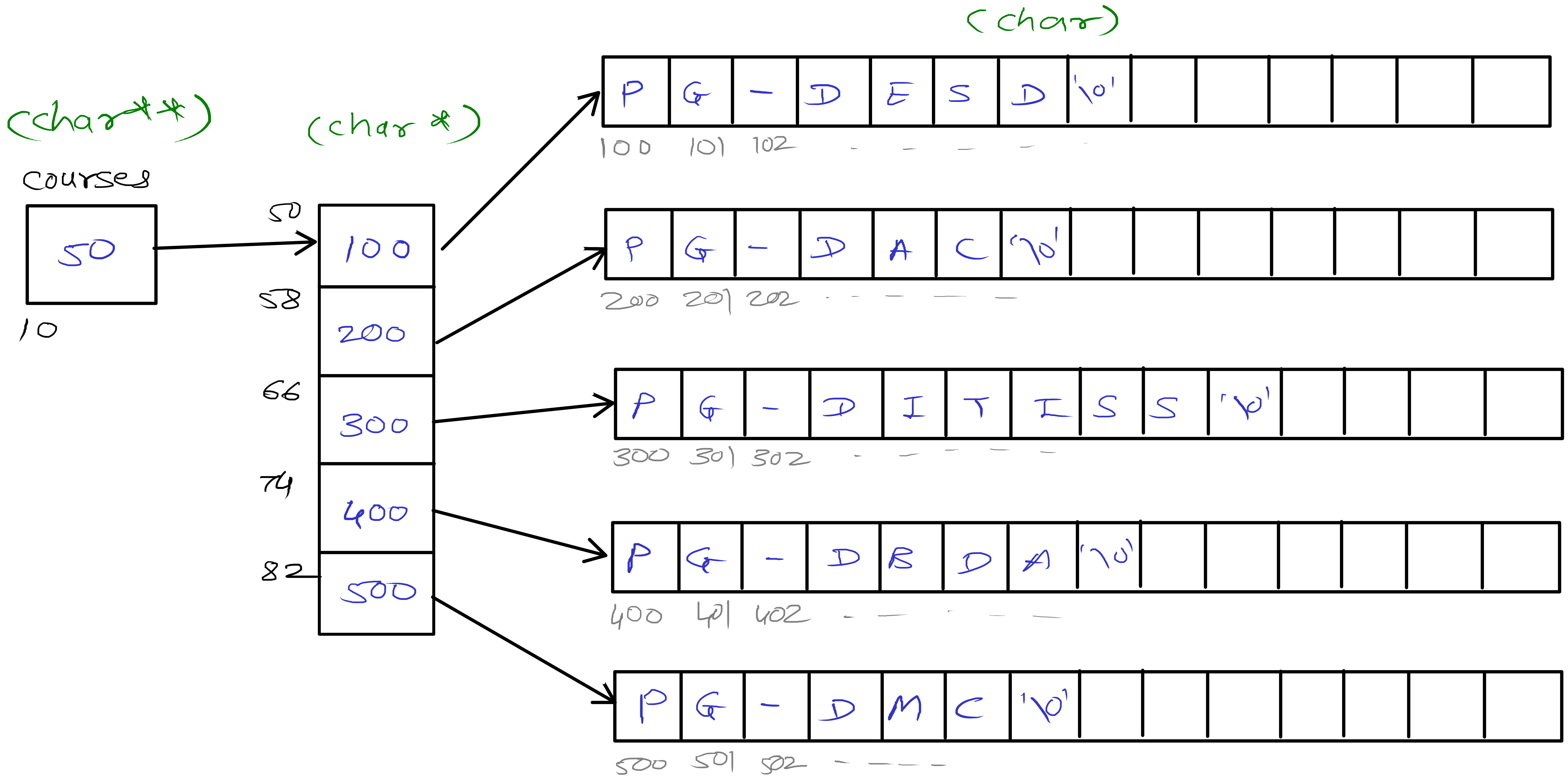
length=20

```
char *name = (char*)malloc(length*sizeof(char));
                    or
char *name = (char*)calloc(length , sizeof(char));
```

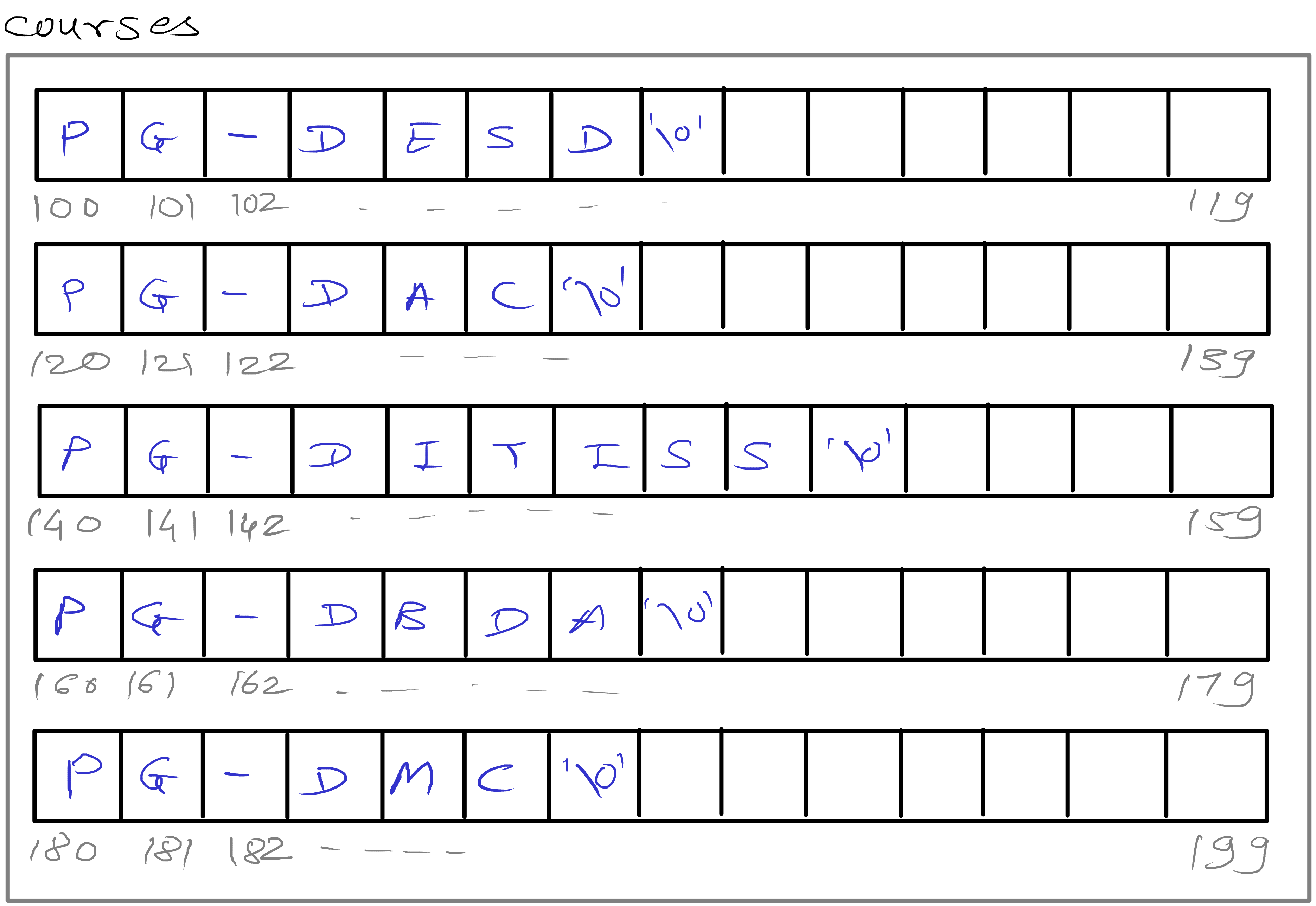


# 2D Array



```
char **courses = (char **)malloc(5 * sizeof(char *));
for(int i =0 ; i < 5 ; i++ )
      courses[i] = (char*) malloc( 20 * sizeof(char));
                                    for(int i =0 ; i < 5 ; i++ )
                                         free(courses[i]);
                                    free(courses);
```

char courses[5][20] = { { { ;



courses → 100
(base addr)

courses[0] → 100
(base addr of first 1D array)

courses[1] → 120
(base addr of second 1D array)

courses[i]
↳ base addr of (i+1)th 1D array.

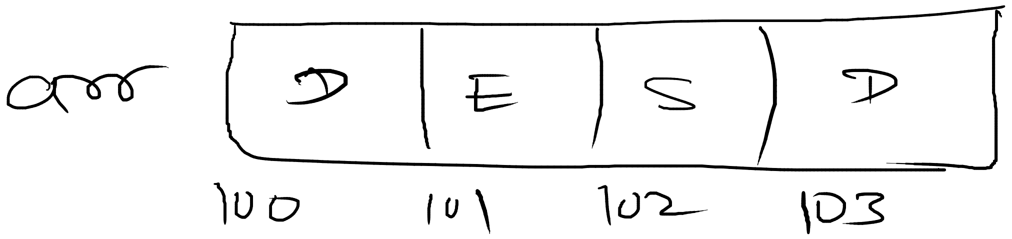
courses[i][j] →
ith row jth col charact

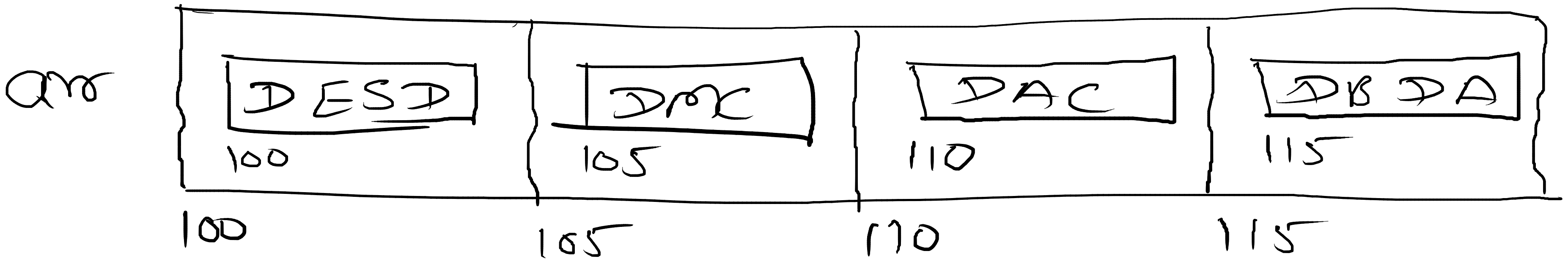
courses → 100 - base addr of 2D array (addr of first 1D array)

*courses → 100 - base addr of 1D Array

courses+1 → 120 - addr of 2nd 1D array.

*(courses+1) → 120 - base addr of 2nd 1D array.





arr → 100 base add$^{r}$

arr → 100 add$^{r}$ of first element

arr+1 → 101 add$^{r}$ of 2nd element

arr+2 → 102 add$^{r}$ of 3rd element

arr+3 → 103 add$^{r}$ of 4th element

*arr = D

*(arr+1) = E

*(arr+2) = S

*(arr+3) = D

arr → 100 base add$^{r}$ of 2D array

arr → 100 add$^{r}$ of first 1D array

arr+1 → 105 add$^{r}$ of 2nd 1D array

arr+2 → 110 add$^{r}$ of 3rd 1D array

arr+3 → 115 add$^{r}$ of 4th 1D array

*arr → 100 base add$^{r}$ of 1st 1D arr

*(arr+1) → 105 base add$^{r}$ of 2nd 1D a

*(arr+2) → 110 base add$^{r}$ of 3rd 1D a

*(arr+3) → 115 base add$^{r}$ of 4th 1D a

*(arr+1)+1 → 106

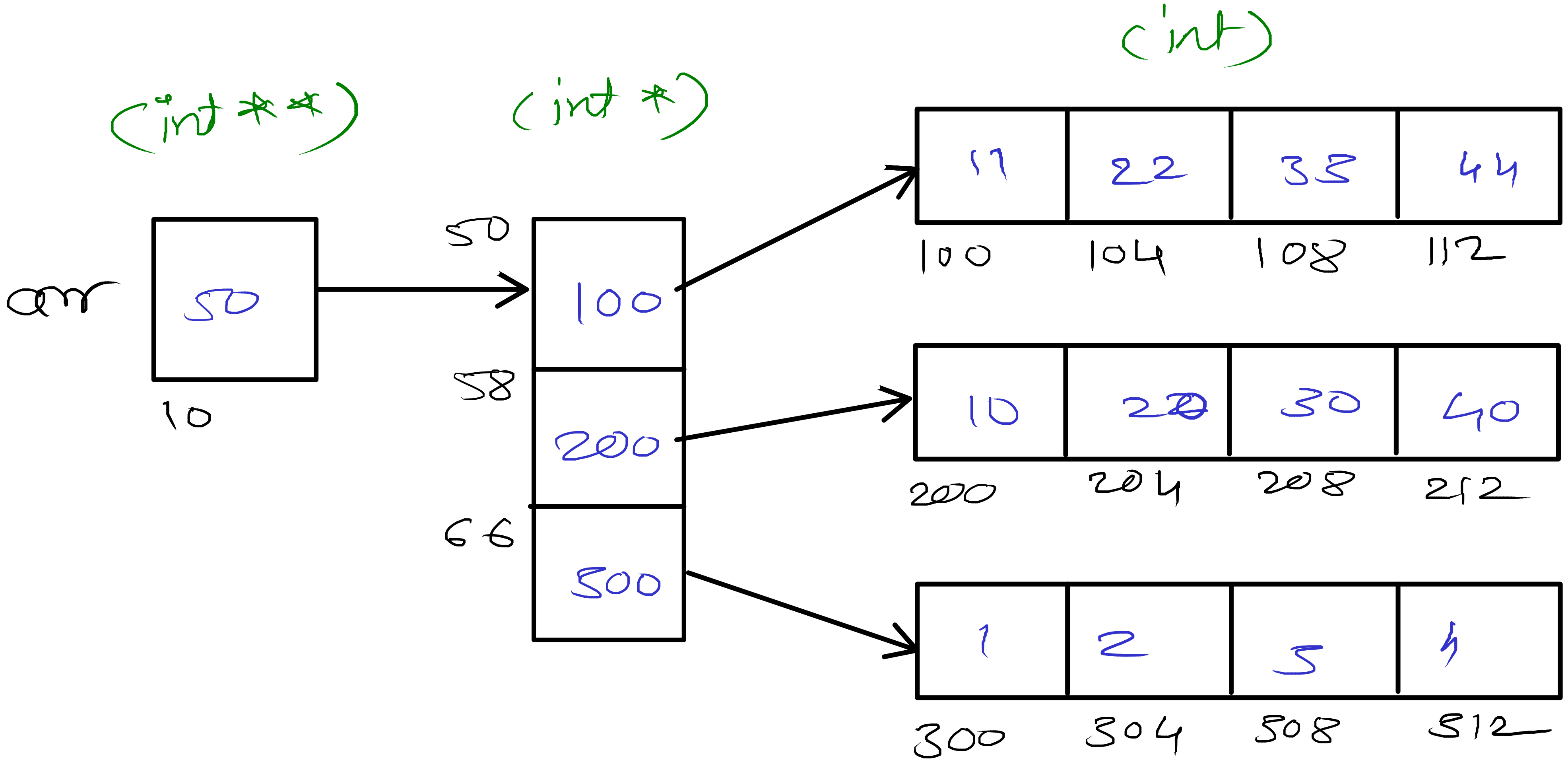
*(arr+i) => arr[i] - base add$^{r}$ of 1D

*(*(arr+i)+j) => arr[i][j]

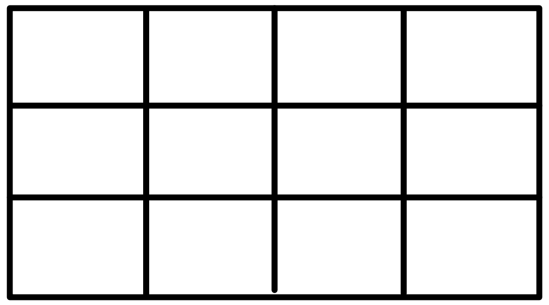
i → 0,1,2,3

j → 0,1,2,3,4

# 2 D Array of integers (3 X 4)



int arr[3][4];



arr = 50
arr[0] = 100
arr[1] = 200
arr[2] = 300

arr[0][0] = 11
arr[0][2] = 33
arr[1][3] = 40
arr[2][1] = 2

```
int **arr = (int **)malloc( 3 * sizeof(int*));
for(int i =0; i< 3 , i++)
    arr[i] = (int *)malloc( 4* sizeof(int));

// - . -

for(int i=0; i< 3; i++)
    free(arr[i]);
free(arr);
```

arr = 50
*arr = 100
*arr+1 = 104
*(arr+1) = 200

3×4 = 12 elements

```
int *ptr = (int *) malloc(12 *sizeof(int));
```

- array of integer (12)

```
int ** ptr = (int **) malloc(12*sizeof(int));
```

- arry of integer pointers (6)

```
char* ptr = (char*) malloc (12 * sizeof(int));
```

- arry of characters (48)

```
char **ptr = (char**) malloc(12 *sizeof(int));
```

- array of character pointers (6)

```
int* arr[3];
  for(i=0; i<3; i++)
      arr[i] = (int *)malloc(4 * sizeof(int));
```

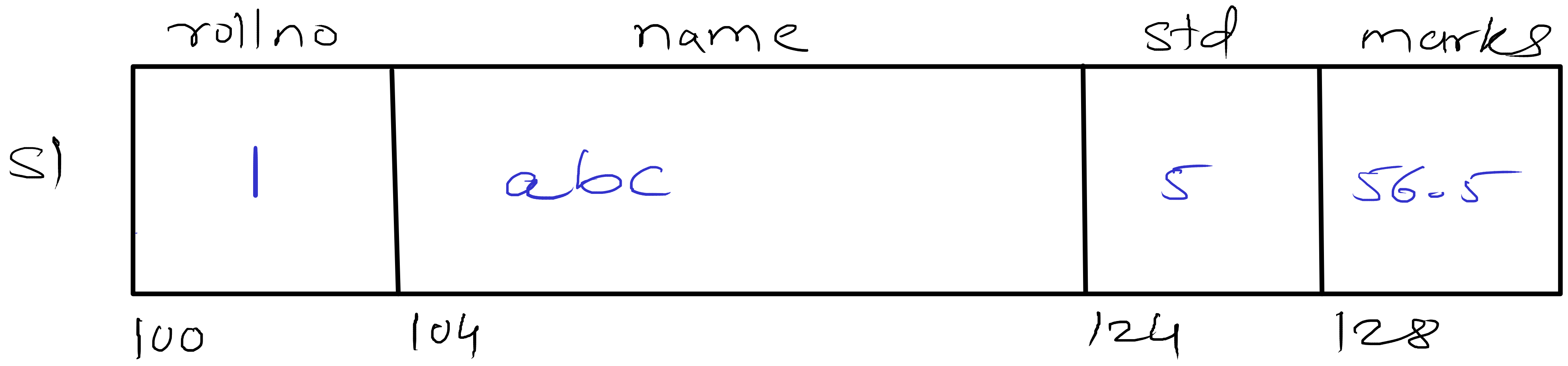
```
int main(void)
{
        ptr=malloc(20);
        //------
        int num=10;
        ptr=&num;
        //------
        return 0;
}
```

if dynamically allocated memory is not reachable due to some reson & is also not able free, then it is leaked. This is known as memory leackage

```
int main(void)
{
        ptr=malloc(20);
        //---
        //---
        free(ptr);
        ptr=2.  ← still addr is
                       present
        return 0;
}
```

pointer which has addr of invalid memory, is known as dangling pointer.

**struct student s1 = {1, "abc", 5, 56.5f};**



To access members —> <name of variable>.<member>

s1.rollno —> 1

s1.name —> "abc"

s1.std —> 5

s1.marks —> 56.5