



Dynamic Memory Allocation

Problem with Arrays

- Sometimes
 - Amount of data cannot be predicted beforehand
 - Number of data items keeps changing during program execution
- Example: Search for an element in an array of N elements
- One solution: find the maximum possible value of N and allocate an array of N elements
 - Wasteful of memory space, as N may be much smaller in some executions
 - Example: maximum value of N may be 10,000, but a particular run may need to search only among 100 elements
 - Using array of size 10,000 always wastes memory in most cases

Better Solution

- Dynamic memory allocation
 - Know how much memory is needed after the program is run
 - Example: ask the user to enter from keyboard
 - Dynamically allocate only the amount of memory needed
- C provides functions to dynamically allocate memory
 - `malloc`, `calloc`, `realloc`

Memory Allocation Functions

- **malloc**

- Allocates requested number of bytes and returns a pointer to the first byte of the allocated space

- **calloc**

- Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.

- **free**

- Frees previously allocated space.

- **realloc**

- Modifies the size of previously allocated space.

- We will only do **malloc** and **free**

Allocating a Block of Memory

- A block of memory can be allocated using the function `malloc`
 - Reserves a block of memory of specified size and returns a pointer of type `void`
 - The return pointer can be type-casted to any pointer type
- General format:

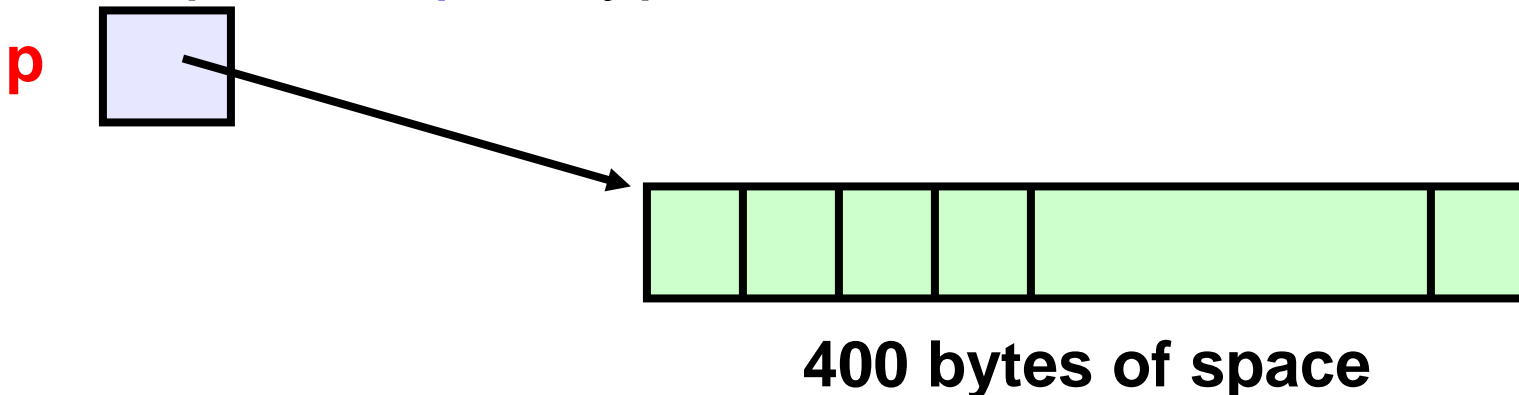
```
type *p;
```

```
p = (type *) malloc (byte_size);
```

Example

```
p = (int *) malloc(100 * sizeof(int));
```

- A memory space equivalent to **100 times the size of an int** bytes is reserved
- The address of the first byte of the allocated memory is assigned to the pointer **p** of type **int**



Contd.

- `cptr = (char *) malloc (20*sizeof(char));`

Allocates 20 bytes of space for the pointer `cptr` of type `char`

- `sptr = (struct stud *) malloc(10*sizeof(struct stud));`

Allocates space for a structure array of 10 elements. `sptr` points to a structure element of type `struct stud`

Always use sizeof operator to find number of bytes for a data type, as it can vary from machine to machine ₇

Points to Note

- `malloc` always allocates a block of contiguous bytes
 - The allocation can fail if sufficient contiguous memory space is not available
 - If it fails, `malloc` returns `NULL`

```
if ((p = (int *) malloc(100 * sizeof(int))) == NULL)
{
    printf ("\n Memory cannot be allocated");
    exit();
}
```

Using the malloc'd Array

- Once the memory is allocated, it can be used with pointers, or with array notation
- Example:

```
int *p, n, i;  
scanf("%d", &n);  
p = (int *) malloc (n * sizeof(int));  
for (i=0; i<n; ++i)  
    scanf("%d", &p[i]);
```

The n integers allocated can be accessed as *p, *(p+1), *(p+2), ..., *(p+n-1) or just as p[0], p[1], p[2], ..., p[n-1]

Example

```
int main()
{
    int i,N;
    float *height;
    float sum=0,avg;

    printf("Input no. of students\n");
    scanf("%d", &N);

    height = (float *)
        malloc(N * sizeof(float));
```

```
    printf("Input heights for %d
students \n",N);
    for (i=0; i<N; i++)
        scanf ("%f", &height[i]);

    for(i=0;i<N;i++)
        sum += height[i];

    avg = sum / (float) N;

    printf("Average height = %f \n",
        avg);
    free (height);
    return 0;
}
```

Releasing the Allocated Space:

`free`

- An allocated block can be returned to the system for future use by using the `free` function
- General syntax:
`free (ptr);`
where `ptr` is a pointer to a memory block which has been previously created using `malloc`
- Note that no size needs to be mentioned for the allocated block, the system remembers it for each pointer returned

Can we allocate only arrays?

- malloc can be used to allocate memory for single variables also
 - `p = (int *) malloc (sizeof(int));`
 - Allocates space for a single int, which can be accessed as `*p`
- Single variable allocations are just special case of array allocations
 - Array with only one element

malloc()-ing array of structures

```
typedef struct{
    char name[20];
    int roll;
    float SGPA[8], CGPA;
} person;
int main() {
    person *student;
    int i,j,n;
    scanf("%d", &n);
    student = (person *)malloc(n*sizeof(person));
    for (i=0; i<n; i++) {
        scanf("%s", student[i].name);
        scanf("%d", &student[i].roll);
        for(j=0;j<8;j++) scanf("%f", &student[i].SGPA[j]);
        scanf("%f", &student[i].CGPA);
    }
    return 0;
}
```

Static array of pointers

```
#define N 20
#define M 10
int main()
{
    char word[N], *w[M];
    int i, n;
    scanf("%d",&n);
    for (i=0; i<n; ++i) {
        scanf("%s", word);
        w[i] = (char *) malloc ((strlen(word)+1)*sizeof(char));
        strcpy (w[i], word) ;
    }
    for (i=0; i<n; i++) printf("w[%d] = %s \n",i,w[i]);
    return 0;
}
```

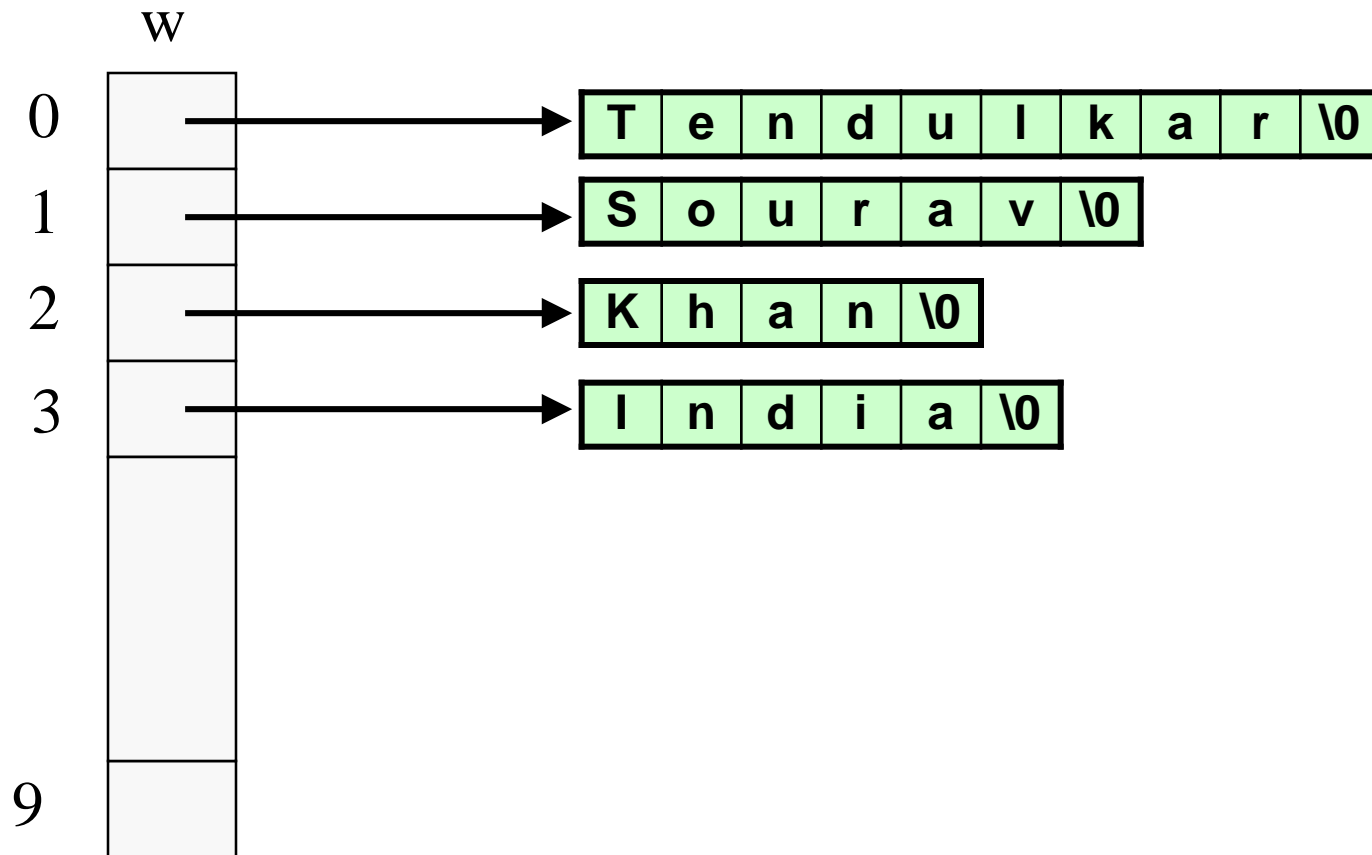
Static array of pointers

```
#define N 20
#define M 10
int main()
{
    char word[N], *w[M];
    int i, n;
    scanf("%d",&n);
    for (i=0; i<n; ++i) {
        scanf("%s", word);
        w[i] = (char *) malloc ((strlen(word)+1)*sizeof(char));
        strcpy (w[i], word) ;
    }
    for (i=0; i<n; i++) printf("w[%d] = %s \n",i,w[i]);
    return 0;
}
```

Output

```
4
Tendulkar
Sourav
Khan
India
w[0] = Tendulkar
w[1] = Sourav
w[2] = Khan
w[3] = India
```

How it will look like



Pointers to Pointers

- Pointers are also variables (storing addresses), so they have a memory location, so they also have an address
- Pointer to pointer – stores the address of a pointer variable

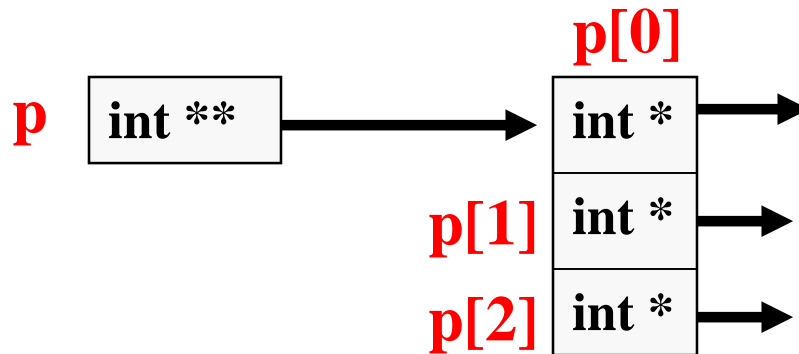
```
int x = 10, *p, **q;  
p = &x;  
q = &p;  
printf(“%d %d %d”, x, *p, *(*q));
```

will print 10 10 10 (since *q = p)

Allocating Pointer to Pointer

```
int **p;
```

```
p = (int **) malloc(3 * sizeof(int *));
```



Dynamic Arrays of pointers



```
int main()
{
    char word[20], **w; /* **w is a pointer to a pointer array */
    int i, n;
    scanf("%d",&n);
    w = (char **) malloc (n * sizeof(char *));
    for (i=0; i<n; ++i) {
        scanf("%s", word);
        w[i] = (char *) malloc ((strlen(word)+1)*sizeof(char));
        strcpy (w[i], word) ;
    }
    for (i=0; i<n; i++) printf("w[%d] = %s \n",i, w[i]);
    return 0;
}
```

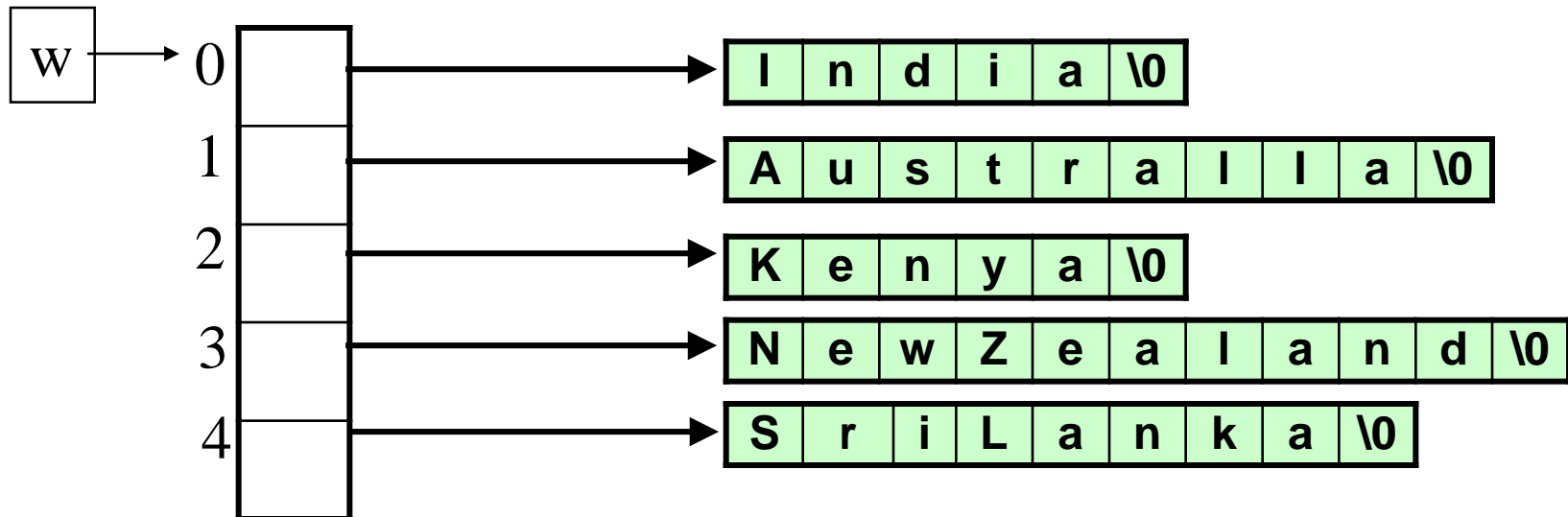
Dynamic Arrays of pointers

```
int main()
{
    char word[20], **w; /* “**w” is a pointer to a pointer array */
    int i, n;
    scanf("%d",&n);
    w = (char **) malloc (n * sizeof(char *));
    for (i=0; i<n; ++i) {
        scanf("%s", word);
        w[i] = (char *) malloc ((strlen(word)+1)*sizeof(char));
        strcpy (w[i], word) ;
    }
    for (i=0; i<n; i++) printf("w[%d] = %s \n",i, w[i]);
    return 0;
}
```

Output

```
5
India
Australia
Kenya
NewZealand
SriLanka
w[0] = India
w[1] = Australia
w[2] = Kenya
w[3] = NewZealand
w[4] = SriLanka
```

How this will look like





Practice Problems

- Take any of the problems you have done so far using 1-d arrays. Now do them by allocating the arrays dynamically first instead of declaring them statically