Dynamic Memory Allocation

- There is a way of allocating memory to a program during **runtime**.
- This is known as dynamic memory allocation.
- Dynamic allocation is done in a part of the memory called the heap.
- You can control the memory allocated depending on the actual input(s)
  - Less wastage

Robotics Club IITKanpur

Malloc - **memory** **alloc**ation

We tell malloc how many bytes are required

malloc allocates those many **consecutive** bytes

Returns the address of (a pointer to) the first byte

Warning: allocated bytes filled with garbage

Warning: if insufficient memory, NULL pointer returned

malloc has no idea if we are allocating an array of floats or chars – returns a void* pointer – typecast it yourself

The allocated memory can be used safely as an array

Robotics Club IITKanpur

malloc: Example

A pointer to float

```
float *f;
f = (float *) malloc(10 * sizeof(float));
```

Size big enough to hold 10 floats.

Note the use of **sizeof** to keep it machine independent

Explicit type casting to convey users intent

malloc evaluates its arguments at *runtime* to allocate (reserve) space. Returns a **void***, pointer to first address of allocated space.

malloc: Example

Key Point: The size argument can be a variable or non-constant expression!

```
float *f; int n;
scanf(“%d”, &n);
f = (float*) malloc(n * sizeof(float));

f[0] = 0.52;
scanf(“%f”, &f[3]); //Overflow if n<=3
printf(“%f”, *f + f[0]);
```

After memory is allocated, pointer variable behaves as if it is an **array**!

This is because, in C, f[i] simply means *(f+i).

Robotics Club IITKanpur

free()

Releasing unused memory is as important as creating it
To release allocated memory use

free()

Deallocates memory allocated by malloc().
Takes a pointer as an argument.

e.g.
free(newPtr);

Freeing unused memory is a good idea, but it's not mandatory. When your program exits, any memory which it has allocated but not freed will be automatically released.

Calloc – **c**ontiguous **alloc**ation

A helpful version of malloc that initializes memory to 0 .

However, slower than malloc since time spent initializing

Use this if you actually want zero initialization

Syntax a bit different – instead of total number of bytes, we need to send it two things

- length of array (number of elements in the array)
- number of bytes per element

Sends back a NULL pointer if insufficient memory – careful!

Need to typecast the pointer returned by calloc too!

calloc() example

```
/* Using calloc() to initialize 100 floats to 0.0 */
#include <stdlib.h>
#include <stdio.h>
#define BUFFER_SIZE 100

int main(){
  float * buffer;
  int i;
  if ((buffer = (float *)calloc(BUFFER_SIZE,sizeof(float))) == NULL)
     {
       printf("out of memory\n");
       exit(1);
     }
  for (i=0; i < BUFFER_SIZE; i++)
       printf(“buffer[%d] = %f\n”, i, buffer[i]);
  return 0;
}
```

realloc()

- If you find you did not allocate enough space use realloc()
- You give realloc() a pointer (such as you received from an initial call to malloc()) and a new size, and realloc does what it can to give you a block of memory big enough to hold the new size

```
int *ip;

ip = (int *)malloc(100 * sizeof(int));
...
/* need twice as much space */
ip = (int *)realloc(ip, 200 * sizeof(int));
```

Robotics Club IITKanpur