EEM103
Computer Programming

Week11

- Passing pointers to function (Call by reference)
- Passing arrays to function
- Dynamic memory allocation

1

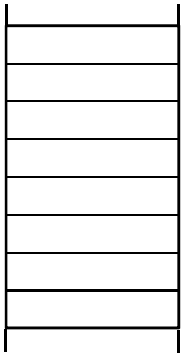
Passing pointers to function (Call by reference)

- There are two ways to pass arguments to a function
  - pass-by-value,
  - pass-by-reference.
- Reference type is a C++ feature, in C, pointers and the indirection operator are used to simulate pass-by-reference.
- When a function argument is defined as a pointer;
  - A local address variable which is belong to function is defined.
  - But it points a local variable of the main (or caller function, in general)

2

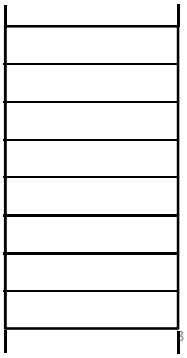
Call by value

```
int main() {
   int a = 5 ;
   func(a) ;
   return 0;
}

void func(int a) {
   a = 10 ;
}
```



Call by reference(in C)

```
int main() {
   int a = 5 ;
   func(&a) ;
   return 0;
}

void func(int *a) {
   *a = 10 ;
}
```



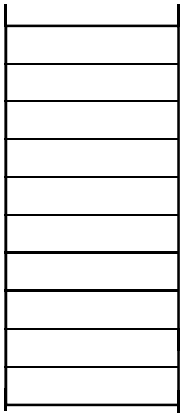
3

Passing arrays to function

- Arrays can also be passed to functions.
- Since array name is a pointer, it is identical to calling by reference.

4

Passing arrays to function



```
void func(int, int [], int);

int main() {
   int a = 5 ;
   int d[4] = {1,2,3,4} ;

   func(a, d, 4);

   printf("a = %d\n\n" , a);
   int i;
   for(i=0; i<4; i++)
      printf("d[%d]=%d \n", i, d[i]);
   return 0;
}

void func(int a, int b[], int n) {
   int i;
   a = a*a ;
   for(i=0; i<n; i++)
        b[i] = b[i]*b[i] ;
}
```

5

- Homework:

How a 2d array is passed to a function?

Find an example.

6

Dynamic memory management

- Ordinary arrays in C are STATIC. That means;
  - They must be defined with a constant array size.
  - Array size is specified at COMPILE TIME and can not be changed.

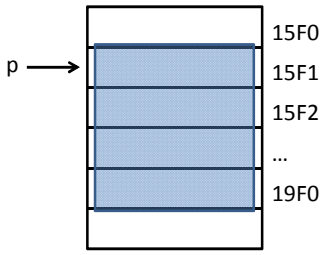
- **Dynamic Memory (Array)** can be defined as;
  - arrays whose size can be a variable,
  - and changed during RUN TIME.
- Dynamic memory functions in C;
  - malloc() realloc()
  - calloc() free()

7

malloc()

- Memory ALLOCation
- Prototype: void* malloc(unsigned int **size**);
  - Allocates **size** Byte in memory.
  - Returns the address of first byte.

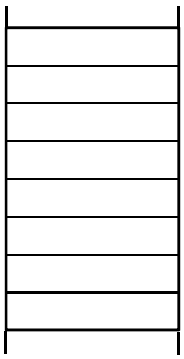
- Eg.: p=malloc(1024);
  - Allocates 1024 Byte,
  - Assigns first address to p.



8

Static Array

```
int main() {
   int d[4] ;

   d[1] = 10 ;
   *(d+2) = 20 ;

   return 0 ;
}
```



Dynamic Array

```
#include "malloc.h"

int main() {
 int N;
 int * d ;

 scanf("%d" , &N);
 d = (int*) malloc( N*sizeof(int)
);
 d[1]  = 10 ;
 *(d+2)= 20 ;

 return 0 ;
}
```

9

malloc()

- If the operating system cannot allocated desired memory, it return 0 (NULL) pointer.
  - This situation should be checked in program..

```
p=malloc(1024);
if(p==NULL)
{
  printf("out of memory");
  exit(1);    /* exit program */
}
```

  - Examples:

p=malloc(10);	10 byte.
p=malloc(10*4);	40 byte.
p=malloc(10*sizeof(int));	10 integer. (40B)
p=malloc(10*sizeof(char));	10 character. (10B)

10

realloc()

- **realloc()** is used to REsize a dynamic memory (that is allocated with malloc)

E.g:

```
p=malloc(100);
…
p=realloc(p, 200);
p=realloc(p, 50);
```

11

free()

- **free()** deletes dynamic memories.

E.g:

```
p=malloc(100);
…
free(p);
```

12

calloc()

- Homework:

Study **calloc()** function.

13

Dynamic Memory allocation in C++

```
#include <iostream>
using namespace std;

int main () {

   int* a ;

   a  = new int;

   *a = 10 ;


   ...

   delete a ;

}
```

```
#include <iostream>
using namespace std;

int main () {

   int* d ;

   d  = new int[4];

   d[1] = 10 ;
   *(d+2) = 20 ;

   ...

   delete [] d ;

}
```

14