# CS 332/532 Systems Programming

Lecture 4

- Loops, Arrays, Pointers -

Professor : Mahmut Unan – UAB CS

# Agenda

- Loops
- Arrays
- 2D Arrays
- Pointers

# for loop

```c
#include <stdio.h>
int main() {
 for (;;){
     printf("This is a strange infinite loop");
 }
}
```

# while loop

```
#include <stdio.h>
int main(void)
{
        int i = 10;
        while(i != 0)
        {
                printf("%d\n", i);
                i--;
        }
        return 0;
}
```

# do-while loop

```
#include <stdio.h>
int main(void)
{
        int i = 1;
        do
        {
                printf("%d\n", i);
                i++;
        } while(i <= 10);
        return 0;
}
```

# Arrays

- **One-Dimensional Arrays**
  - An array is a data structure that contains a number of values, or else elements, of the same type.
  - Each element can be accessed by its position within the array
  - Always declare the array before your try to use them

    **data_type** array_name[number_of_elements];

```
int sampleArray[100];
float anotherArray[250];
```

# predefined size

```
/* use macros */
#define ARRAY_SIZE 250
float sampleArray[ARRAY_SIZE];
```

```
/* never use const */
const int array_size = 250;
float sampleArray(array_size)
/* this is not legal */
```

# sizeof()

```c
#include <stdio.h>
int main() {

    printf("%lu\n", sizeof(char));
    printf("%lu\n", sizeof(int));
    printf("%lu\n", sizeof(float));
    printf("%lu\n", sizeof(double));

    int a = 25;
    double d= 40.55;
    printf("%lu\n",sizeof(a+d));

    int arr[10] = {5,8,9,12};
    printf("\n Size of the array :%lu", sizeof(arr));
    printf("\n Capacity the array :%lu", sizeof(arr)/sizeof(arr[0]));

    int arr2[] = {5,8,9,12};
    printf("\n Size of the array2 :%lu", sizeof(arr2));
    printf("\n Capacity the array2 :%lu", sizeof(arr2)/sizeof(arr2[0]));
    return 0;
}
```

# sizeof()

```c
#include <stdio.h>
int main() {

    printf("%lu\n", sizeof(char));    1
    printf("%lu\n", sizeof(int));     4
    printf("%lu\n", sizeof(float));   4
    printf("%lu\n", sizeof(double));  8

    int a = 25;
    double d= 40.55;
    printf("%lu\n",sizeof(a+d));   8

    int arr[10] = {5,8,9,12};
    printf("\n Size of the array :%lu", sizeof(arr));                    40
    printf("\n Capacity the array :%lu", sizeof(arr)/sizeof(arr[0]));    10

    int arr2[] = {5,8,9,12};
    printf("\n Size of the array2 :%lu", sizeof(arr2));                  16
    printf("\n Capacity the array2 :%lu", sizeof(arr2)/sizeof(arr2[0])); 4
    return 0;
}
```

# Initialize the Array

```
int arr[3]={10,20,30};
int arr2[10]={10,20};

int arr3[]={10,20,30,40};

/* be careful with the
following*/
const int arr4[] = {10,20,30,40}
```

# Assign - Access elements

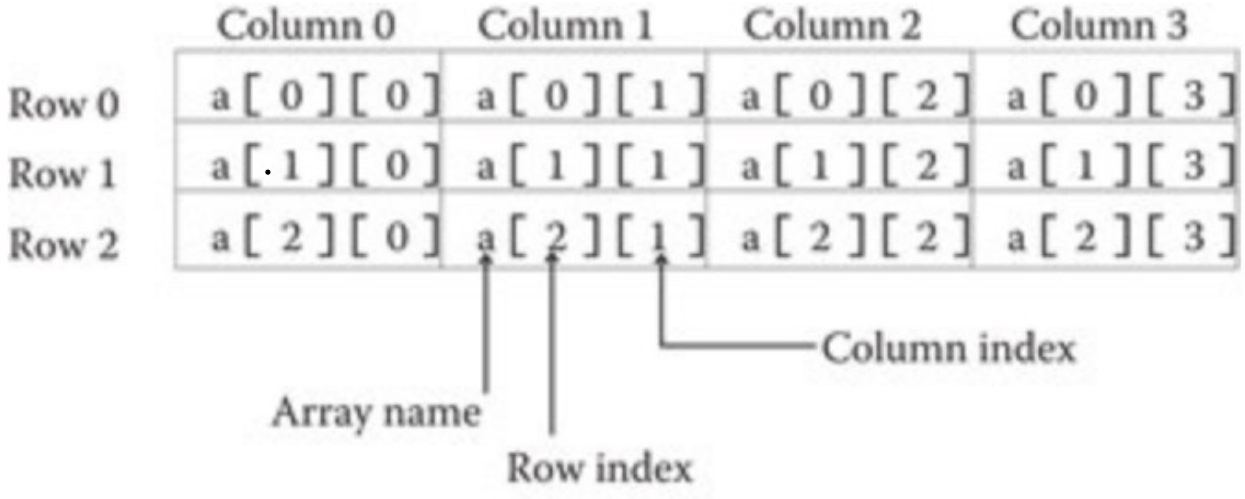
```c
#include <stdio.h>
int main() {
    int i,j=10, arr[10];
    arr[0]=10;
    arr[1]=arr[0]*2;
    for (i=2;i<10;i++){
        arr[i]=j*(i+1);
    }
    for (i=0;i<10;i++)
        printf("\n arr[%d] :%d",i,arr[i]);

    return 0;
}
```

```
arr[0] :10
arr[1] :20
arr[2] :30
arr[3] :40
arr[4] :50
arr[5] :60
arr[6] :70
arr[7] :80
arr[8] :90
arr[9] :100
```

# 2D Arrays

- **data_type** array_name[number_of_rows][number_of_columns]

```
int a[3][4];
```



# initialize 2D array

```
int arr[3][3] = {{10, 20, 30},
{40, 50, 60}, {70, 80, 90}};


int arr[3][4] = {10, 20, 30, 40,
50, 60, 70, 80, 90};
```

```
10 20 30 40
50 60 70 80
90 0   0  0
```

# Pointers

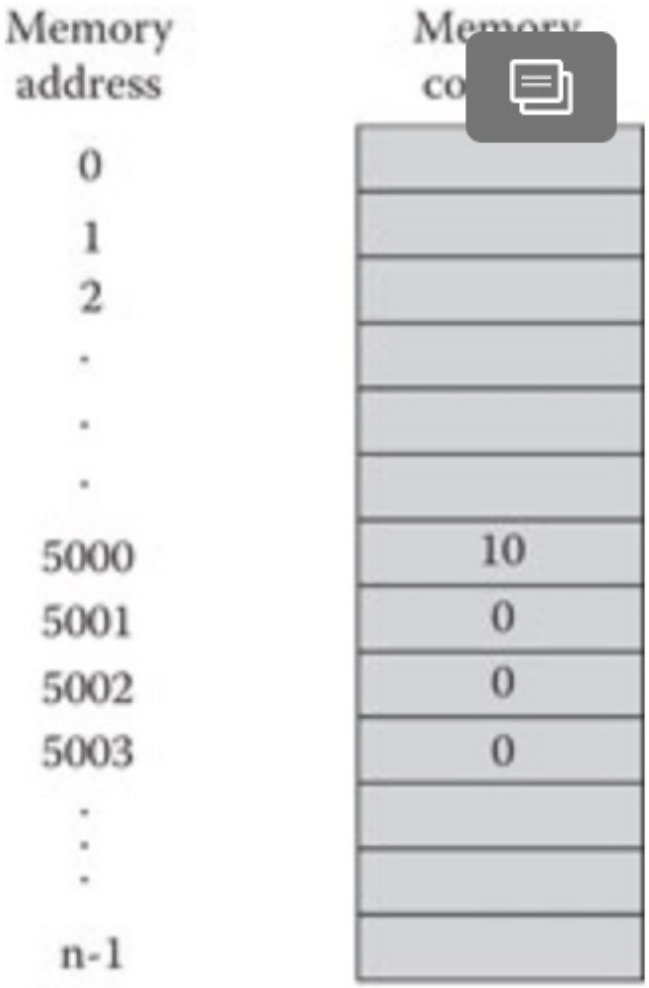
```
int a = 5;
float arr[25];
```

- /* To reach the memory location variables, arrays…etc use ampersand (&) operator */

```
printf("\n %x", &a);
printf("\n %x", &arr);
```

```
e7ad78b8
e7ad78c0
```

# Memory Address



# Pointers / 2

- How to store this address?
  - we use the pointers
  - Pointers is a variable whose value is the address of another variable
- Declare the pointer before you use it

```
data_type *pointer_name;
int *ptr, a, b, c;
int * ptr, a, b, c;
int* ptr, a, b, c;
```

!!! Caution !!!!

All three statements are correct and the result of each statement the “ptr” will be declared as the pointer but a,b, and c will be declared as int.
However; it is always better to use the first syntax

```
data_t  pointer_name;
int *ptr, a, b, c;
int * ptr, a, b, c;
int* ptr, a, b, c;
```

# size of a pointer ???

```c
#include <stdio.h>
int main() {
    char c;
    char *ptrC = &c;
    int a=5;
    int *ptrI = &a;
    float f =20.66;
    float *ptrF = &f;
    double d = 44.445;
    double *ptrD = &d;

    printf("size of c: %u\n", sizeof(c));
    printf("size of ptrC: %u\n", sizeof(ptrC));
    printf("size of a: %u\n", sizeof(a));
    printf("size of ptrI: %u\n", sizeof(ptrI));
    printf("size of f: %u\n", sizeof(f));
    printf("size of ptrF: %u\n", sizeof(ptrF));
    printf("size of d: %u\n", sizeof(d));
    printf("size of ptrD: %u\n", sizeof(ptrD));
    return 0;
}
```

```
size of c: 1
size of ptrC: 8
size of a: 4
size of ptrI: 8
size of f: 4
size of ptrF: 8
size of d: 8
size of ptrD: 8
```

```c
#include <stdio.h>
int main(void)
{
    int *ptr, a;
    a = 10;
    ptr = &a;
    printf("Val = %d\n", *ptr);
    return 0;
}
```

10

Always initialize the pointer before using it, otherwise you will get segmentation fault error

# Example - page 1/2

```c
#include <stdio.h>
int main()
{
    int *ptr, a;
    a = 25;
    /* without using a pointer */
    printf("Address of a: %p\n", &a);
    printf("Value of a: %d\n", a);

    /*let's use a pointer */
    ptr = &a;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);

    /* how about if we change the value of int */
    a = 125;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);

```

# Example - page 2/2

```
20      /* let's change the value using pointer*/
21      *ptr = 250;
22      printf("Address of a: %p\n", &a);
23      printf("Value of a: %d\n", a);
24
25      /* we can reuse the pointer */
26      int b = 50;
27      ptr = &b;
28      printf("Address of the pointer : %p\n", ptr);
29      printf("Value of the pointer : %d\n", *ptr);
30      return 0;
31  }
```

# Example - output

```
Address of a: 0x7ffeee56291c
Value of a: 25
Address of the pointer : 0x7ffeee56291c
Value of the pointer : 25
Address of the pointer : 0x7ffeee56291c
Value of the pointer : 125
Address of a: 0x7ffeee56291c
Value of a: 250
Address of the pointer : 0x7ffeee562918
Value of the pointer : 50
```

# References


- https://www.tutorialspoint.com/cprogramming/c_constants.htm
- C From Theory to Practice - 2nd edition, Nikolaos D. Tselikas and George S. Tselikis