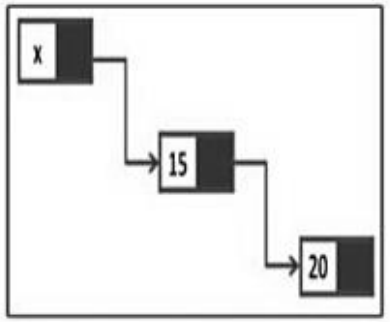
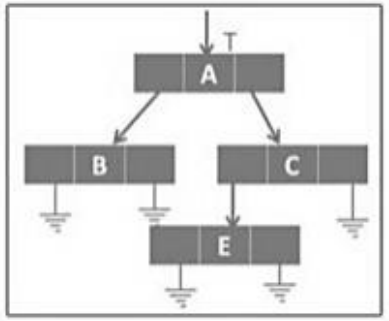
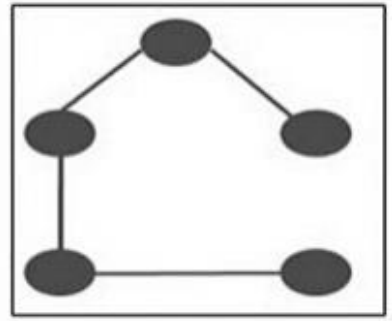
Sorting



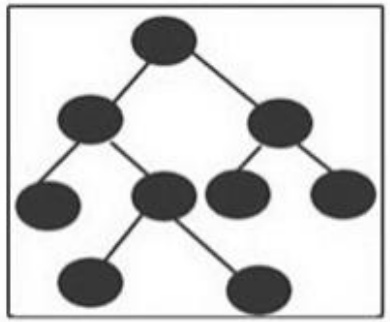
Link list



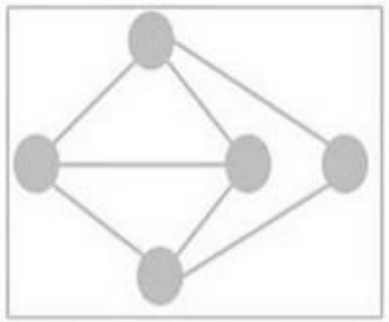
list



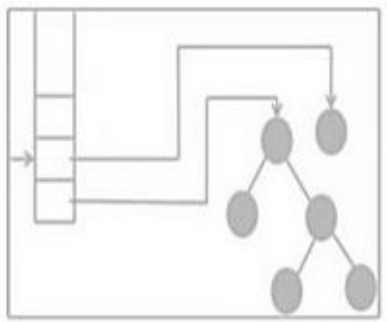
spanning tree



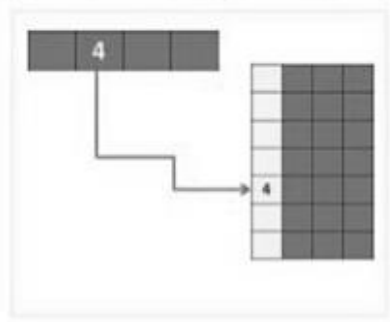
Tree



Graph



Stack



Hashing

CS 2124: Data Structures
Spring 2024

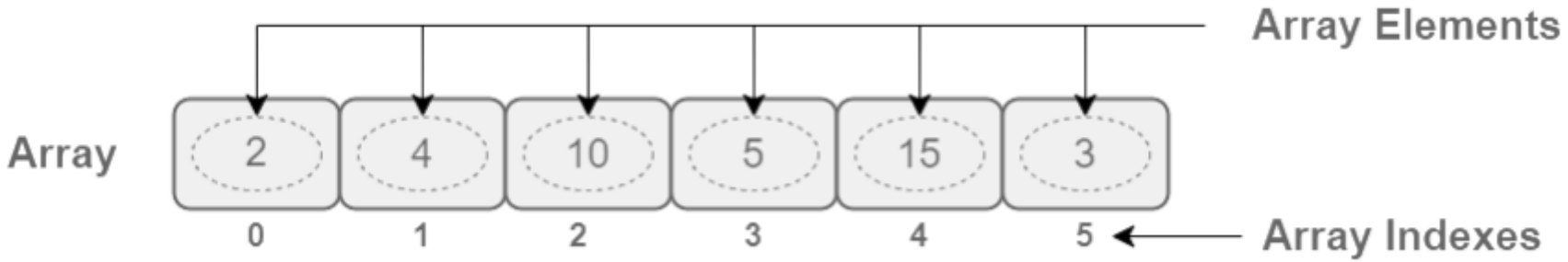
First Lecture Part –II

Topics

- Arrays
- Pointers
- Multidimensional Array
  - Applications of 3D arrays
- Structure and Union

Array (Revision)

- An array is a collection of items of same data type stored at contiguous memory locations.
- The idea is to store multiple items of the same type together.
- This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).



Array (Revision)
Applications

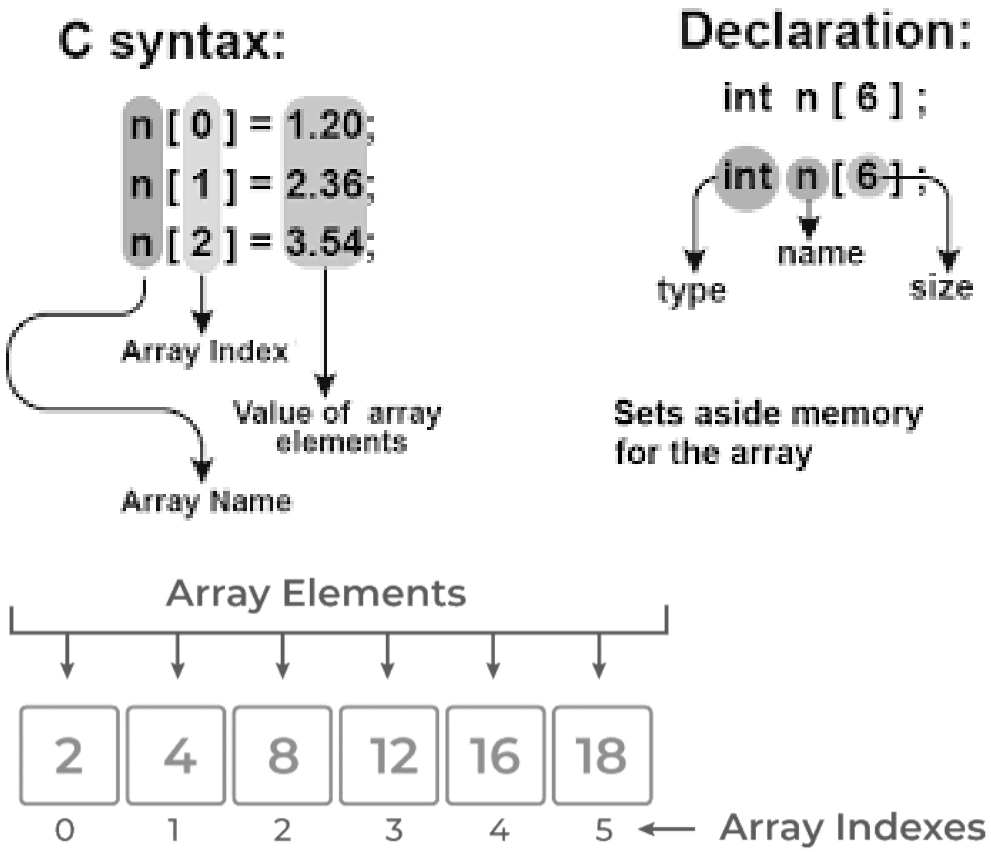
- Array provide easy access to all the elements at once and the order of accessing any element does not matter.

  - Arrays can be used for sorting data elements.
  - Arrays can be used for performing matrix operations.
  - Arrays can be used for CPU scheduling.
  - Arrays are used in signal processing to represent a set of samples that are collected over time.
  - Arrays are used in robotics to represent the position and orientation of objects in 3D space.
  - Arrays are used in real-time monitoring and control systems to store sensor data and control signals

Array (Revision)

You have already studied about arrays and are well-versed with the techniques to utilize these data structures

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

   int  x[5];
   int j;
   for(j=0; j<5; j++)
   {
    printf("Array Position:%d \n", j);
    x[j] = 2*j;
    printf("Value of J: %d \n", x[j]);
}
return 0;
}
```



Array (Revision)

You have already studied about arrays and are well-versed with the techniques to utilize these data structures

```c
#include <stdio.h>
int main ()
{
    int  x[2]={20,24};
    int j;
    for(j=0; j<2; j++)
    {
    printf("Value of x[%d]: %d \n", j, x[j]);
    printf("Add of x[%d]: %p \n", j, &x[j]);
    }
}
```

A. Program will not compile due to error
B. Program will compile but with warning
C. Will compile and run without any error or warning

```c
#include <stdio.h>
int main()
{
     // array initialization using initialier list
    int arr[5] = { 10, 20, 30, 40, 50 };
    printf("Arr: %d", arr[2]);
    // array initialization using initializer list without
    // specifying size
    int arr1[] = { 1, 2, 3, 4, 5 };
    printf("\n Arr1: %d", arr1[2]);
    // array initialization using for loop
    float arr2[5];
    for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;
    }
    printf("\n Arr2:%d", arr2[2]);
    return 0;
}
```

Declaring an Array

- Will this code work ?

a) Warning but will work
b) Error will not compile
c) Will compile and work
d) I know the answer but I am not going to share it with the class

Declaring an Array

Program A

```c
#include <stdio.h>
int main()
{
    float arr2[];
    for (int i = 0; i < 5; i++)
    {
        arr2[i] = (float)i * 2.1;
    }
     for (int i = 0; i < 5; i++)
    printf("Arr2[%d]: %f \n", i, arr2[i]);
}
```

Program B

```c
#include <stdio.h>
int main()
{
    float arr2[5];
    for (int i = 0; i < 5; i++)
    {
        arr2[i] = (float)i * 2.1;
    }
     for (int i = 0; i < 5; i++)
    printf("Arr2[%d]: %f \n", i, arr2[i]);
}
```

A. Both Program will not compile due to error
B. Both Program will compile but with warning
C. Program B Will compile and run without any error or warning but Program A will not compile
D. Program A Will compile and run without any error or warning but Program B will not compile

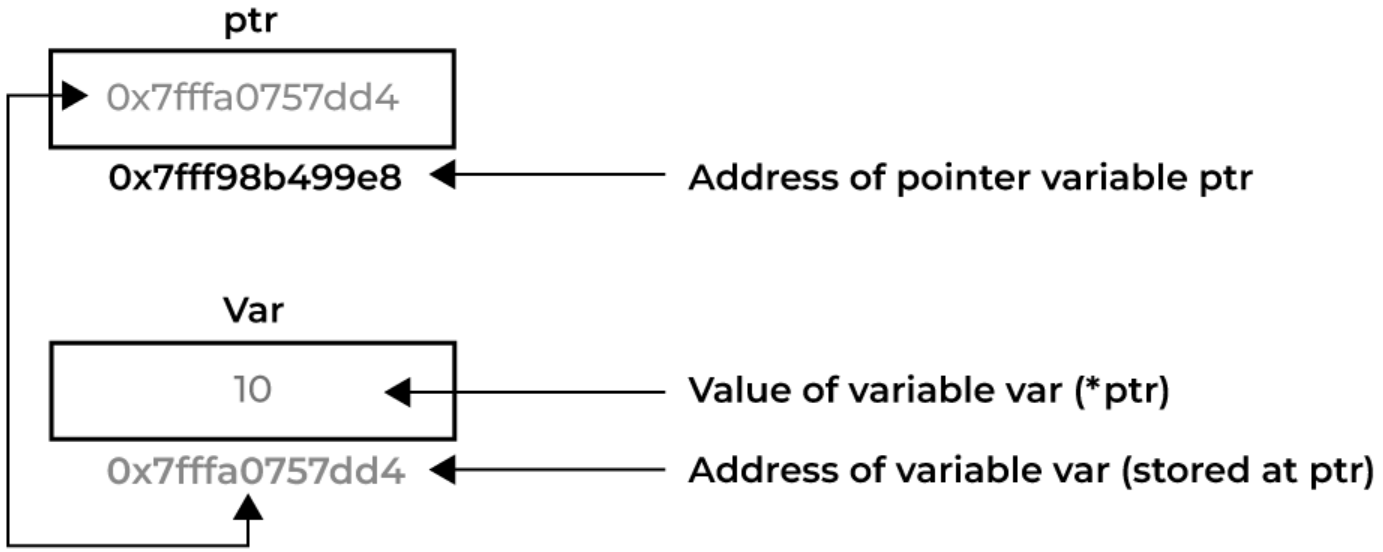
Addresses

```c
#include <stdio.h>
int main()
{
     // array initialization using initialier list
    int arr[5] = { 10, 20, 30, 40, 50 };
    printf("Address Arr[2]: %d ; Arr[3]: %p", &arr[2], &arr[3]);
    // array initialization using initializer list without
    // specifying size
    int arr1[] = { 1, 2, 3, 4, 5 };
    printf("\n Address Arr1[2]: %d ; Arr1[3]: %p", &arr1[2], &arr1[3]);
    // array initialization using for loop
    float arr2[5];
    for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;
    }
    printf("\n Address Arr2[2]: %d ; Arr2[3]: %p", &arr2[2], &arr2[3]);
    return 0;
}
```

A. There will be no warnings on lines 6, 10 and 16
B. Program will not compile due to error
C. Program will compile but with warnings on lines 6, 10 and 16

Pointer (Revision)

- A pointer is defined as a derived data type that can store the address of other C variables or a memory location.
- We can access and manipulate the data stored in that memory location using pointers.
- As the pointers store the memory addresses, their size is independent of the type of data they are pointing to.



The size of the pointer depends on the architecture. However, in **32-bit architecture** the size of a pointer is 2 byte, **64-bit architecture** the size of a pointer is 8 byte

Pointer (Revision)

- Is this going to work ?

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

   int  var1;
   char var2[10];

printf("Address of var1 variable: %p\n", &var1  );
printf("Address of var2 variable: %p\n", &var2  );
printf("Address of var2[0]: %p and var2[9]: %p\n", &var2[0], &var2[9]);
return 0;
}
```

Pointer (Revision)

- Is this going to work ?

```
1  #include <stdio.h>
2  int main () {
3
4     int  var = 20;
5     int  *ip;
6
7     ip = &var;
8
9    printf("1: %p\n", &var  );
10   printf("2: %p\n", ip );
11   printf("3: %d\n", *ip );
12   printf("4: %p\n", &ip );
13    return 0;
14 }
```

1. Program compiles but with warnings
2. Error, will not compile
3. Will compile and run but garbage on some outputs
4. Will compile and run with value 20 as output on line 11. While other outputs (lines 9, 10 and 12) will print addresses

Pointer (Revision)

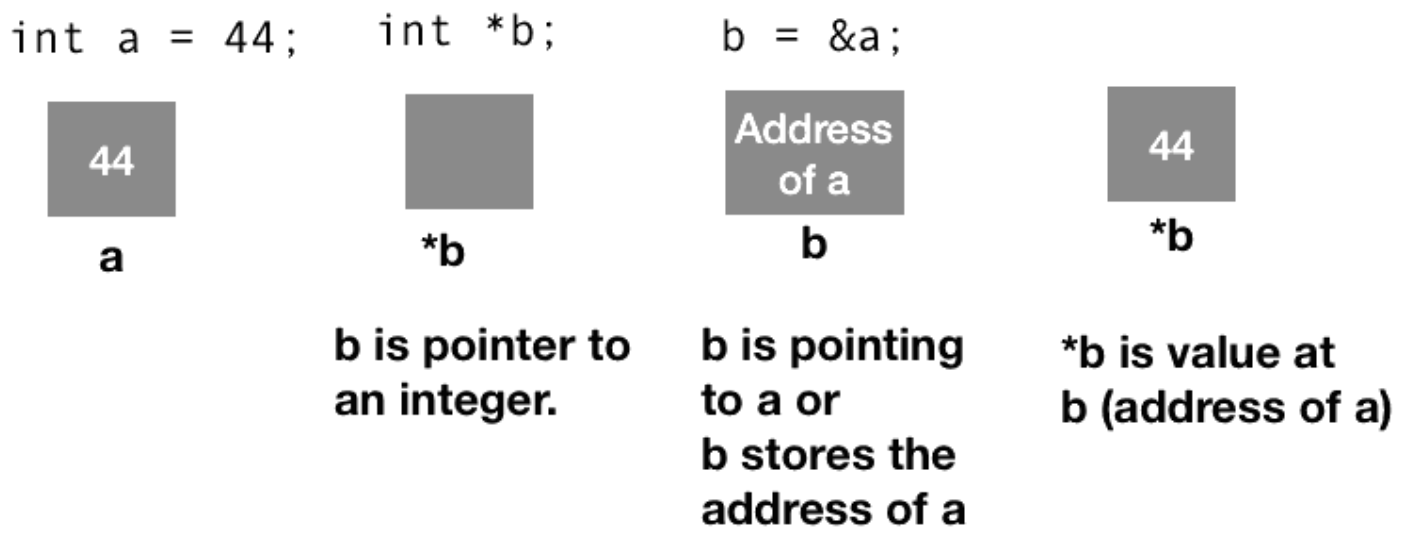
- Is this going to work ?
- Changes for understanding

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

   int  var = 20;
   int* ip;

   ip = &var;

  printf("Address of var variable: %p\n", &var  );
  printf("Address stored in ip variable: %p\n", ip );
  printf("Value of *ip variable: %d\n", *ip );
  printf("Address of *ip variable: %p\n", &ip );
   return 0;
}
```

Pointer (Revision)

- A pointer is a variable whose value is the address of another variable of the same type.

```c
#include <stdio.h>
int main () {
 int *a, *b, c=1, d=2;
 a = &c;
 b = &d;
 printf("1:%d \n", *a);
 printf("2:%d \n", *b);
 printf("3:%d \n", c);
 printf("4:%d \n", d);
}
```



What will be the output of the code ?

Pointer (Revision)

- int age = 39; // Variable declaration
- int* ptr = &age; // Pointer declaration can also be like **int *p1; int * p2;**
  - **“int* p1, p2;”** p1 is a pointer while p2 is an integer variable

- // Reference: Output the memory address of age with the pointer (0x7ffe5367e044)
- printf("%p\n", ptr);

- // De-reference: Output the value of age with the pointer (39)
- printf("%d\n", *ptr);

Try to implement the code

Pointer (Revision)

```c
#include <stdio.h>
int main()
{
int c=1, *pc;
// both &c and pc are addresses
pc = &c;  //error or not
printf("1: %p \n", pc);
// both c and *pc are values
*pc = c;  //error or not
printf("2: %d \n", *pc);
// pc is address but c is not
pc = c;  //error or not
printf("3: %d \n", pc);
// &c is address but *pc is not
*pc = &c; //error or not
printf("4: %p", pc);
}
```

Pointers and Arrays

```c
#include <stdio.h>
int main()
{
int a[]={1, 2, 3, 4, 5};
int i;
for(i=0; i<5; i++)
printf("Array: %d ; Value: %d ; Something: %d \n", i, a[i], *(a+i) );
return 0;
}
```

- a[ ]={1, 2, 3, 4, 5};
- As we saw ‘a’ is the array address i.e. slide 13

Try to implement the code

Pointers and Arrays

```c
#include <stdio.h>
int main()
{
int a[]={1, 2, 3, 4, 5};
int i;
for(i=0; i<5; i++)
printf("Array: %d ; Value: %d ; Something: %d \n", i, a[i], *(a+i) );
return 0;
}
```

```c
#include <stdio.h>
int main()
{
int a[]={1, 2, 3, 4, 5};
int i;
for(i=0; i<5; i++)
printf("Array: %d ; Value: %d ; Something: %d \n", i, a[i], *(a+i) );
printf("Something: %d \n", *(a) );
printf("Something: %d \n", *(a+1) );
printf("Something: %d \n", *(a+2) );
return 0;
}
```

Try to implement the code

Multidimensional Array

- Multidimension arrays can be defined as ‘Array of arrays that stores similar data in tabular form.’
- Multidimensional arrays are in row-major array.
- Generally they are declared as follows:
  - <Data_type> <array_name>[size1][size2]…[sizenN]
  - Int a[2][3]

- Size of multidimensional array can be calculated by multiplying the dimensions of the array.
  - Int a[2][3] = It can store 2*3 = 6 elements
  - Int a[2][3][3] = it can store 2*3*3= 18 elements

  - int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} } →

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

One and Two dimensional (2D) Arrays

1 D ARRAY:

C	O	D	I	N	G	E	E	K
0	1	2	3	4	5	6	7	8

← single row of elements

2 D ARRAY:

i \ j	col 0	col 1	col 2
row 0	A	A	A
row 1	B	B	B
row 2	C	C	C

← column

} array elements

↑ rows

X[3][3] =

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

1-D array

```
arr = [1, 2, 3, 4, 5]
```

(0)

1	2	3	4	5

arr[0], arr[1], arr[2]

2-D array

```
arr = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
      ]
```

(0, 0)

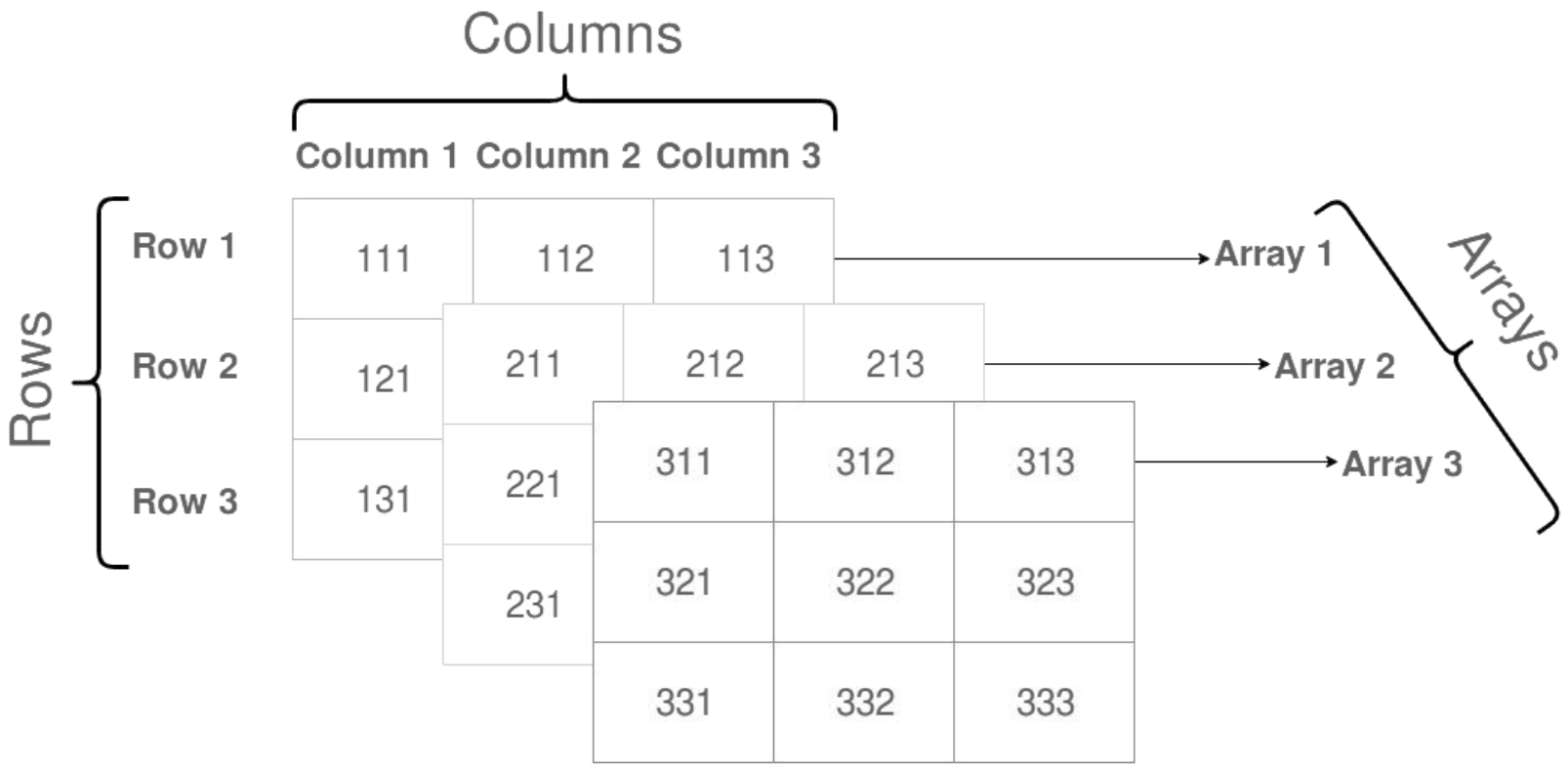
1	2	3
4	5	6
7	8	9

arr[0][0] => 1
arr[1][2] => 6
arr[2][0] => 7

Two dimensional (2D) Arrays

```c
#include <stdio.h>

int main()
{
int x[3][4] = {0, 1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11};

 for (int i = 0; i < 3; ++i)
  {
    for (int j = 0; j < 4; ++j)
    {
      printf("X [%d][%d]= %d \n",i,j, x[i][j]);
    }
  }
  return 0;
}
```

Multidimensional Array



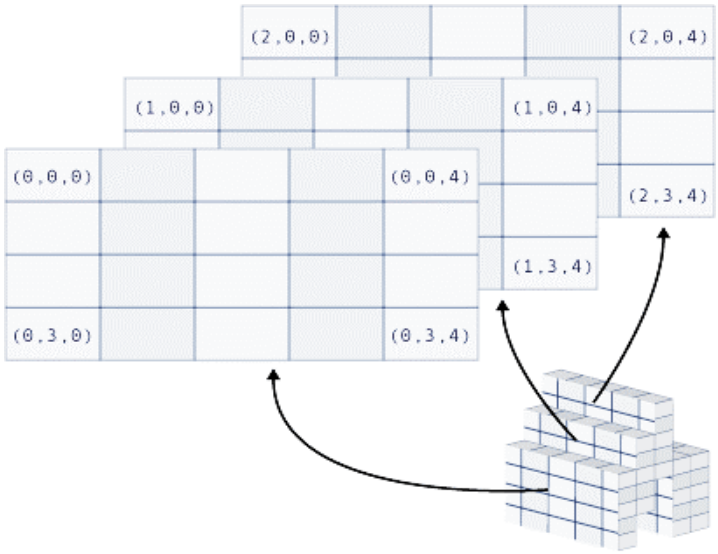
a[x][y][z]

- **x:** Index of 2D array.
- **y:** Index of that 2D array row.
- **z:** Index of that 2D array column.

Image source: Link

Multidimensional Array

```c
#include <stdio.h>
int main()
{
  int test[2][2][2] = {{{3, 4}, {0, -3}}, {{23, 12},{13, 4}}};
    int i,j,k;
  printf("\nDisplaying values:\n");
  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 2; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        printf("test[%d][%d][%d] = %d\n", i, j, k, test[i][j][k]);
      }
    }
  }
  return 0;
}
```



Try to implement the code

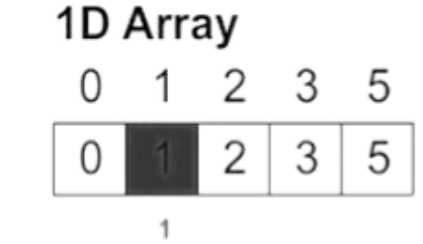
Three Dimensional (3D) arrays

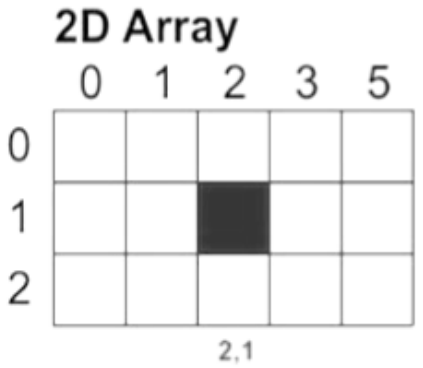
```c
#include <stdio.h>

int main()
{
int x[2][2][2] =
{
    {{3, 4 }, {0, -3 }},
    {{13, 4 }, {5, 9 }}

};

 for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 2; ++j)
    {
        for (int k = 0; k < 2; ++k)
    {
      printf("X [%d][%d][%d]= %d \n",i,j,k, x[i][j][k]);
    }
    }
  }
  return 0;
}
```

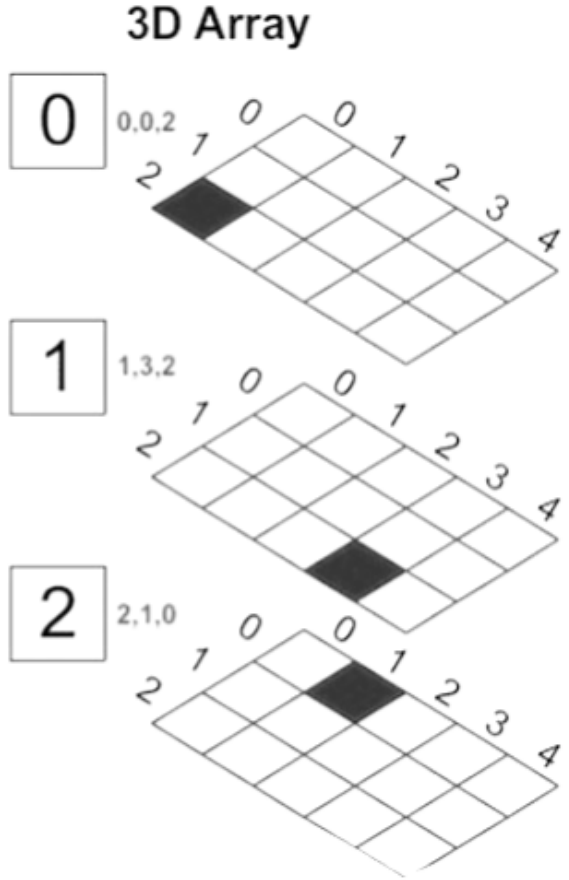
Applications of 3D arrays

- Can be used to implement data structures such as

1. List
2. Stack
3. Queues
4. Trees
5. Graph







Structures & Union

- **Structures:** Collection of one or more variables (possibly of different data types, grouped together under a single name for convenient handling)

- **Unions** are declared in the same fashion as structs, but have a fundamental difference. Only one item within the union can be used at any time, because the memory allocated for each item inside the union is in a shared memory location.

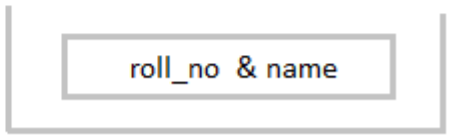
Structure

```
struct Student
{
  int roll_no;        // size 4 byte
  char name[40]; // size 40 byte
} s;
```

size of **s** will 44 bytes

Union

```
union Student
{
  int roll_no;        // size 4 byte
  char name[40]; // size 40 byte
} s;
```



size of **s** will 40 bytes

Structure

```
struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
```

size of e1= 1 + 2 + 4 = 7

Union

```
union Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 4 byte
```

size of e1= 4 (maximum size of 1 element)

Structures & Union

- **Advantages of union**
  - It occupies less memory compared to structure.
  - When you use union, only the last variable can be directly accessed.
  - Union is used when you have to use the same memory location for two or more data members.
  - It enables you to hold data of only one data member.
  - Its allocated space is equal to maximum size of the data member.

- **Advantages of structure**
  - Structures gather more than one piece of data about the same subject together in the same place.
  - It is helpful when you want to gather the data of similar data types and parameters like first name, last name, etc.
  - It is very easy to maintain as we can represent the whole record by using a single name.
  - In structure, we can pass complete set of records to any function using a single parameter.
  - You can use an array of structure to store more records with similar types.

Structures & Union

- **Disadvantages of structure**
  - If the complexity of IT project goes beyond the limit, it becomes hard to manage.
  - Change of one data structure in a code necessitates changes at many other places. Therefore, the changes become hard to track.
  - Structure is slower because it requires storage space for all the data.
  - You can retrieve any member at a time in structure whereas you can access one member at a time in the union.
  - Structure occupies space for each and every member written in inner parameters while union occupies space for a member having the highest size written in inner parameters.
  - Structure supports flexible array. Union does not support a flexible array.
- **Disadvantages of union**
  - You can use only one union member at a time.
  - All the union variables cannot be initialized or used with varying values at a time.
  - Union assigns one common storage space for all its members.

Structures & Union

```c
#include <stdio.h>
#include <string.h>
struct student
{
int rollno;
char name[60];
}s1;  //declaring s1 variable for structure
void main( )
{
//store first employee information
s1.rollno=1;
strcpy(s1.name, "Johnny");//copying string into char array
//printing first employee information
printf( "Rollno : %d\n", s1.rollno);
printf( "Name : %s\n", s1.name);
}
```

Structures & Union

```c
#include <stdio.h>
#include <string.h>
union student
{
int rollno;
char name[60];
}s1;  //declaring s1 variable for union
void main( )
{
//store first employee information
s1.rollno=1;
strcpy(s1.name, "UTSA");//copying string into char array
//printing first employee information
printf( "Rollno : %d\n", s1.rollno);
printf( "Name : %s\n", s1.name);
}
```

```c
#include <stdio.h>
#include <string.h>

struct Books {
   char  title[30];
   char  author[30];
   int   book_id;
};

int main( ) {
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */
   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Author 1");
   Book1.book_id = 1111;
   /* book 2 specification */
   strcpy( Book2.title, "Programming with C");
   strcpy( Book2.author, "Author 2");
   Book2.book_id = 2222;
   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 book_id : %d\n", Book1.book_id);
   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 book_id : %d\n", Book2.book_id);
   return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
 union Data {
   int i;
   float f;
   char str[20];
};
 int main( ) {
   union Data data;
   data.i = 1;
   data.f = 2.5;
   strcpy( data.str, "C Programming");
   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);
   data.i = 10;
   printf( "data.i : %d\n", data.i);
   data.f = 220.5;
   printf( "data.f : %f\n", data.f);
   strcpy( data.str, "C Programming");
   printf( "data.str : %s\n", data.str);
   return 0;
}
```

Try to implement the code

End of Lecture

Do try to implement the codes by your self to better understand the working