UNIT 5 : CONCEPTS OF ARRAYS AND POINTERS

5.1 Concepts of Single-dimensional Array

5.1.1 Numeric single dimensional Array

5.1.2 Numeric single dimensional array operations:

5.1.2.1 Sorting array in ascending or descending. (Bubble and selection)

5.1.2.2 Searching element from array (Linear Search)

5.1.3 Character Single dimensional Array

5.1.3.1 Character Single dimensional array operations:

5.1.3.2 Use of \0, \n and \t

5.2 Pointers:

5.2.1 Concepts of Pointers

5.2.2 Declaring and initializing int, float, char and void pointers

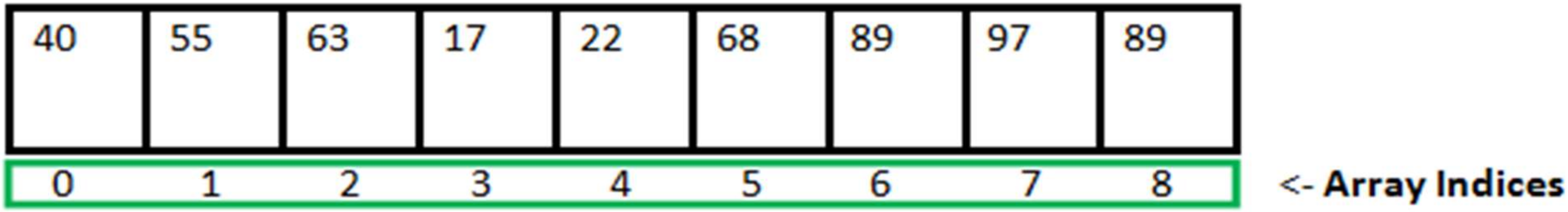
5.2.3 Pointer to single dimensional numeric array

Why do we need arrays?

- We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.

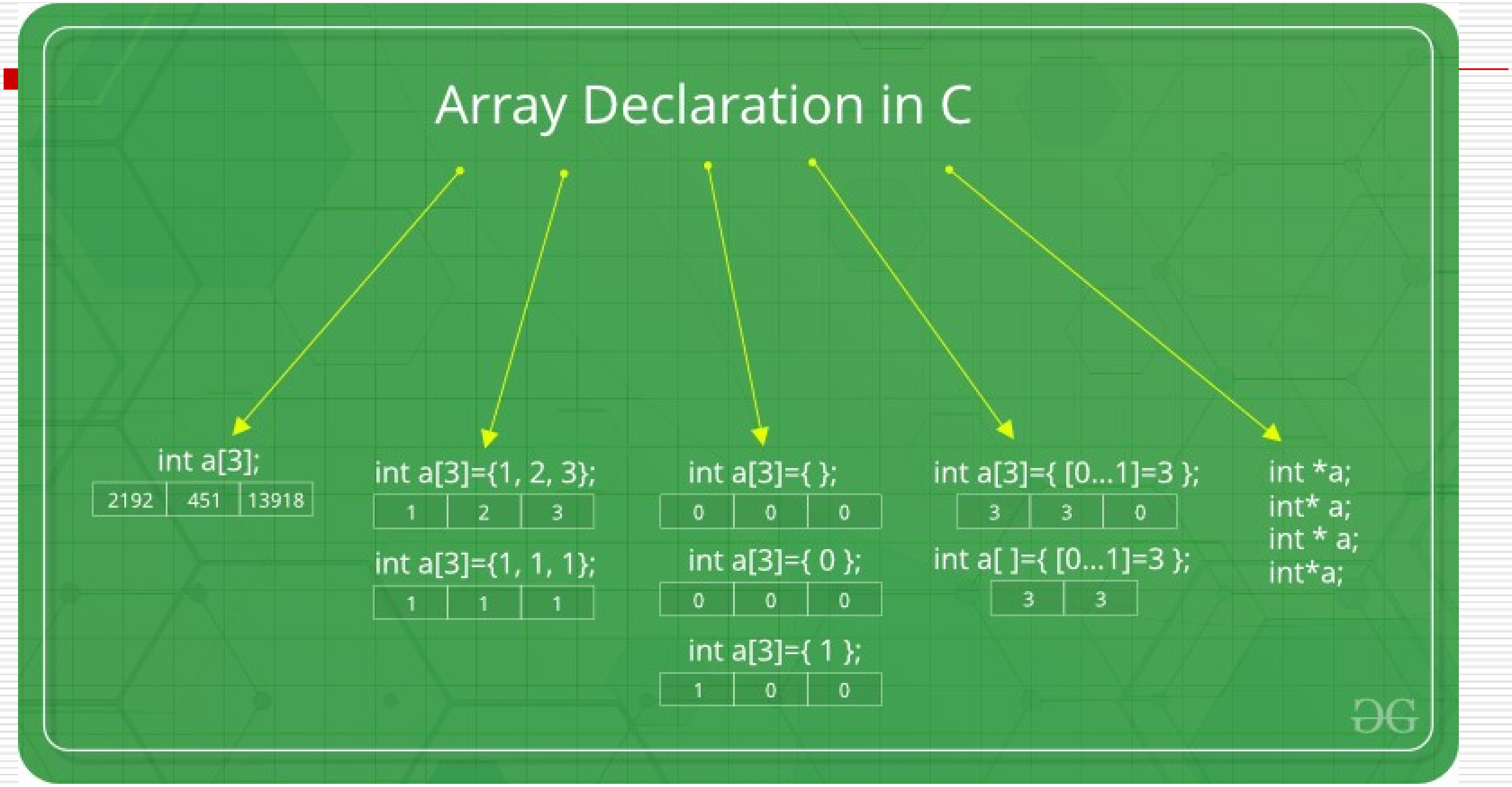
Array

- Indexed collection of similar data type is known as Array.
- Array Index always starts from 0 [ZERO].
- Index can be never Negative.



40	55	63	17	22	68	89	97	89	
0	1	2	3	4	5	6	7	8	**<- Array Indices**

Array Length = 9
First Index = 0
Last Index = 8



Array Declaration in C

```
int a[3];
int a[3]={1, 2, 3};
int a[3]={1, 1, 1};
int a[3]={ };
int a[3]={ 0 };
int a[3]={ 1 };
int a[3]={ [0...1]=3 };
int a[ ]={ [0...1]=3 };
int *a;
int* a;
int * a;
int*a;
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3];// Array Declare
    clrscr();
    a[0]=10;
    a[1]=40;
    a[2]=84;
    printf("%d",a[0]);
    printf("\n%d",a[1]);
    printf("\n%d",a[2]);
    getch();
}
```

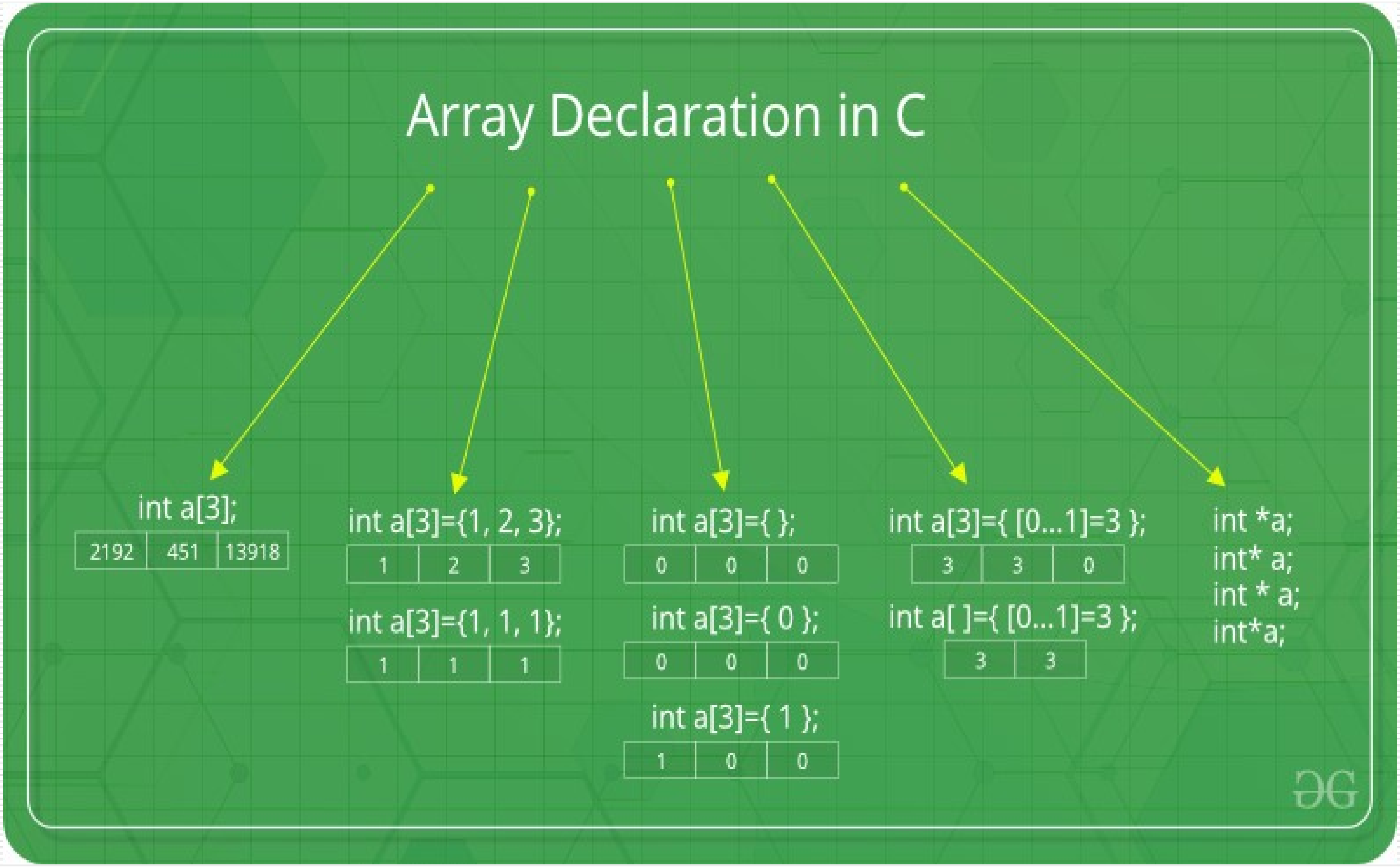
- 5.1 Concepts of Single-dimensional Array
- 5.1.1 Numeric single dimensional Array
- 5.1.2 Numeric single dimensional array operations:
  - 5.1.2.1 Sorting array in ascending or descending. (Bubble and selection)
  - 5.1.2.2 Searching element from array (Linear Search)
- 5.1.3 Character Single dimensional Array
  - 5.1.3.1 Character Single dimensional array operations:
  - 5.1.3.2 Use of \0, \n and \t

- Two types of Array:
  - ***One / Single Dimensional Array (1 D Array)***
  - Two / Double Dimensional Array (2 D Array)
  - Multi Dimensional Array

5.1 Concepts of Single-dimensional Array **OR** Numeric single dimensional Array

- The array which is used to represent and store data in a linear form is called as 'single or one dimensional array.'
- **Syntax:**

  <data-type> <array_name> [size];
- **Example:**
  - int a[3] = {2, 3, 5};
  - int id[5]={1,2,3} ;
  - float tax[3] = {5003.23, 1940.32, 123.20} ;



Array Declaration in C

int a[3];

2192	451	13918

int a[3]={1, 2, 3};

1	2	3

int a[3]={1, 1, 1};

1	1	1

int a[3]={ };

0	0	0

int a[3]={ 0 };

0	0	0

int a[3]={ 1 };

1	0	0

int a[3]={ [0...1]=3 };

3	3	0

int a[ ]={ [0...1]=3 };

3	3

int *a;
int* a;
int * a;
int*a;

Compile Time Initialization

- When we assign value at the time of array declaration is known as Compile Time Initialization.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3]={10,40,85};// Compile Time Initialization
    clrscr();
    printf("%d",a[0]);
    printf("\n%d",a[1]);
    printf("\n%d",a[2]);
    getch();
}
```

Run Time Initialization

- When we assign value at the time of run is known as Run Time Initialization.
- Run Time Initialization is done using scanf().

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[3];// Run Time Initialization
        clrscr();
        scanf("%d",&a[0]);
        scanf("%d",&a[1]);
        scanf("%d",&a[2]);
        clrscr();
        printf("You Have Entered....\n");
        printf("%d",a[0]);
        printf("\n%d",a[1]);
        printf("\n%d",a[2]);
        getch();
}
```

5.1.1 Numeric single dimensional Array

5.1.2 Numeric single dimensional array operations:

5.1.2.1 Sorting array in ascending or descending. (Bubble and selection)

5.1.2.2 Searching element from array (Linear Search)

5.1.2.1 Sorting array in ascending or descending. (Bubble and selection)

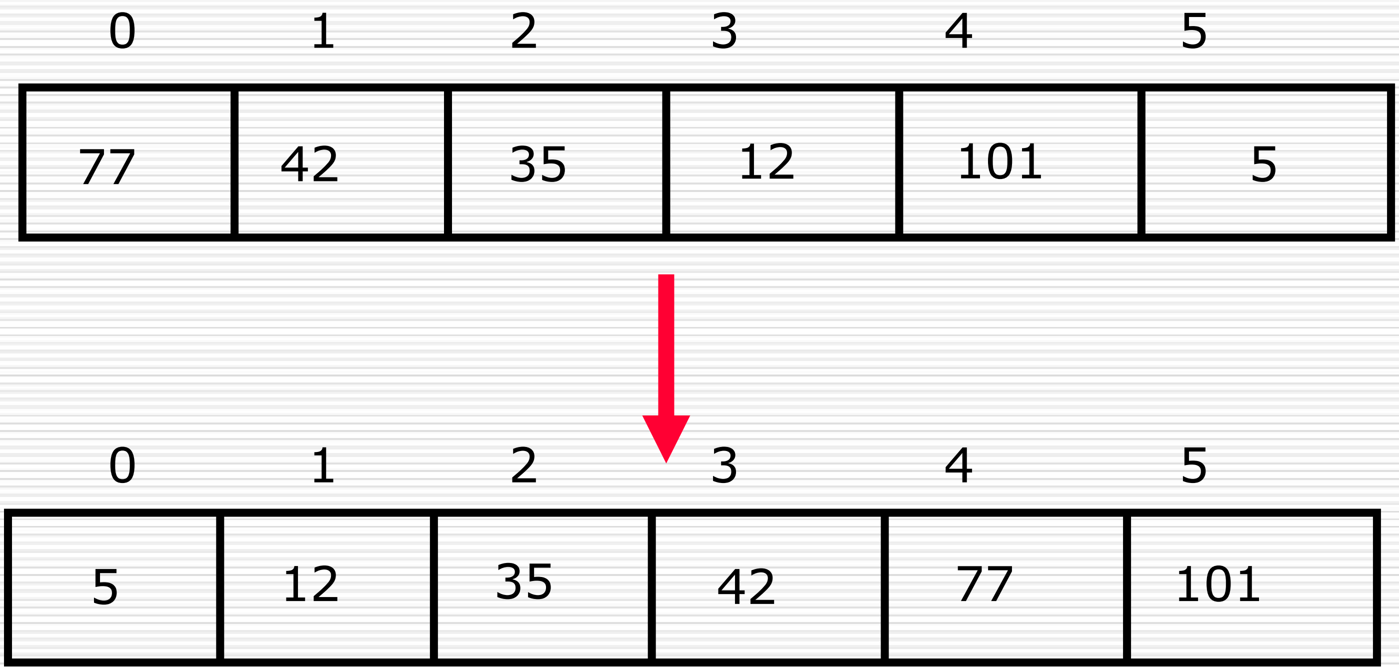
Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

How Bubble Sort works?

Sorting

- □ **Sorting takes an unordered collection and makes it an ordered one.**



0	1	2	3	4	5
77	42	35	12	101	5

0	1	2	3	4	5
5	12	35	42	77	101

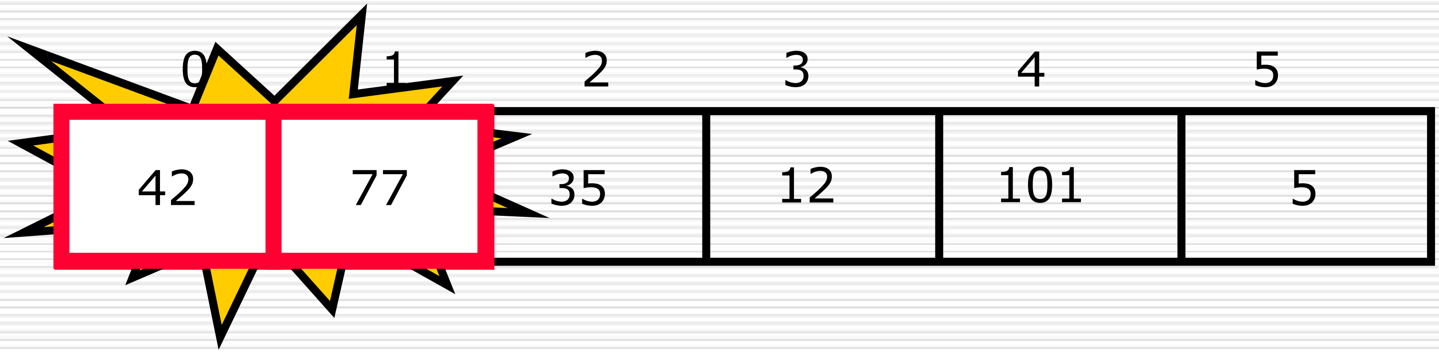
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

0	1	2	3	4	5
77	42	35	12	101	5

"Bubbling Up" the Largest Element

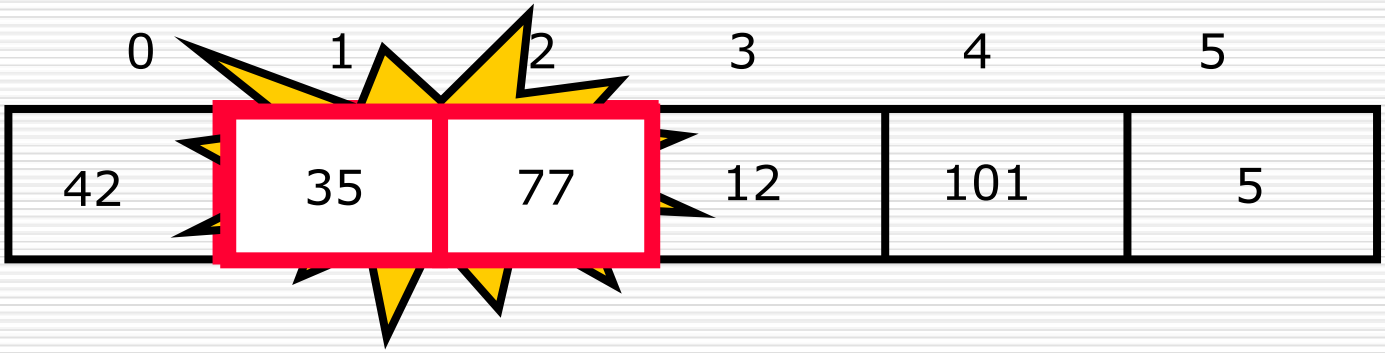
- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



0	1	2	3	4	5
42	77	35	12	101	5

"Bubbling Up" the Largest Element

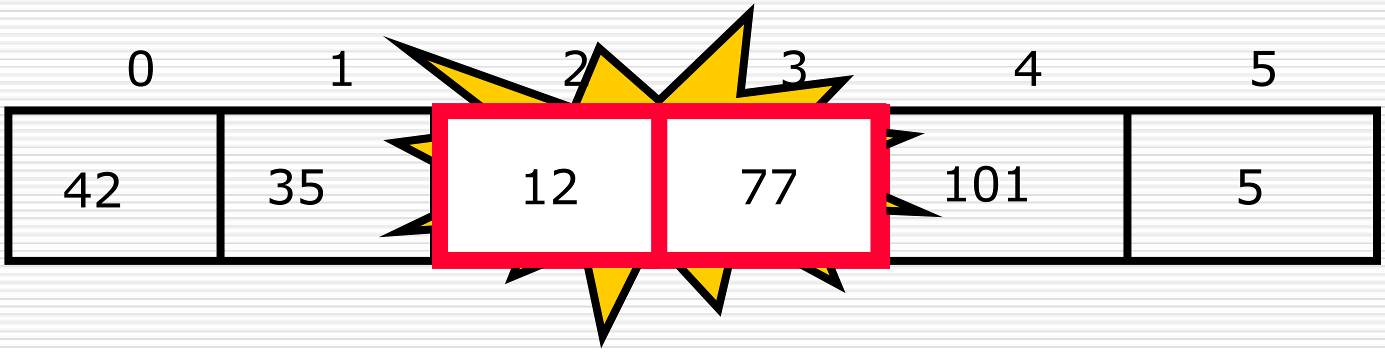
- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



0	1	2	3	4	5
42	35	77	12	101	5

"Bubbling Up" the Largest Element

- □ **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**



0	1	2	3	4	5
42	35	12	77	101	5

"Bubbling Up" the Largest Element

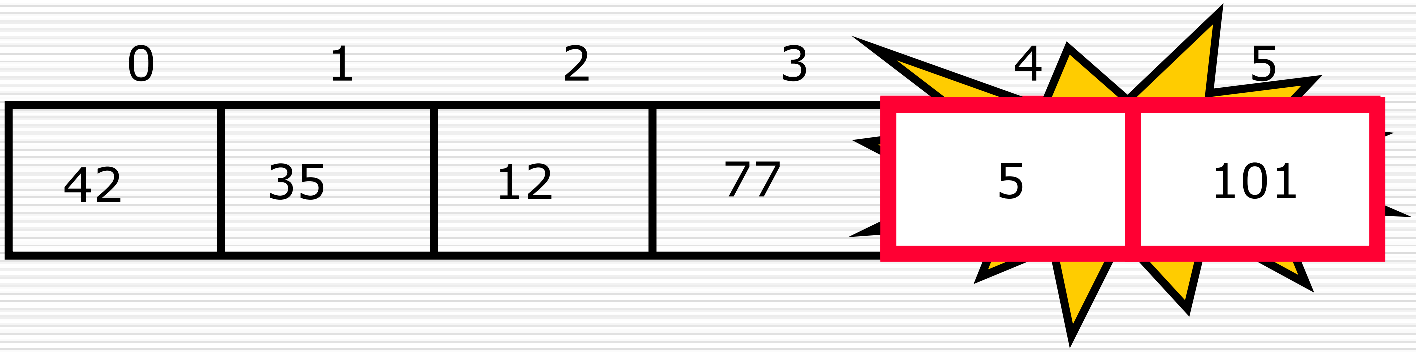
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

0	1	2	3	4	5
42	35	12	77	101	5

No need to swap

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



0	1	2	3	4	5
42	35	12	77	5	101

"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

0	1	2	3	4	5
42	35	12	77	5	101

Largest value correctly placed

- 1 3 5 4 8 9 11
- 20 15 14 16 9 2 1 17

```
// W A P to sort an array with 10 elements using Bubble Sort (Ascending)
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,n[10],temp=0;
        clrscr();
        printf("Enter 10 element :");
        for(i=0;i<=9;i++)
                scanf("%d",&n[i]);
        clrscr();
        printf("your array is :\n");
        for(i=0;i<=9;i++)
                printf("%d\t",n[i]);
        for(i=0;i<=9;i++)
        {
                for(j=0;j<=9;j++)
                {
                                if(n[j]>n[j+1])
                                {
                                                temp=n[j];
                                                n[j]=n[j+1];
                                                n[j+1]=temp;
                                }
                }
        }
        printf("Sorted array is :\n");
        for(i=0;i<=9;i++)
                printf("%d\t",n[i]);

        getch();
}
```

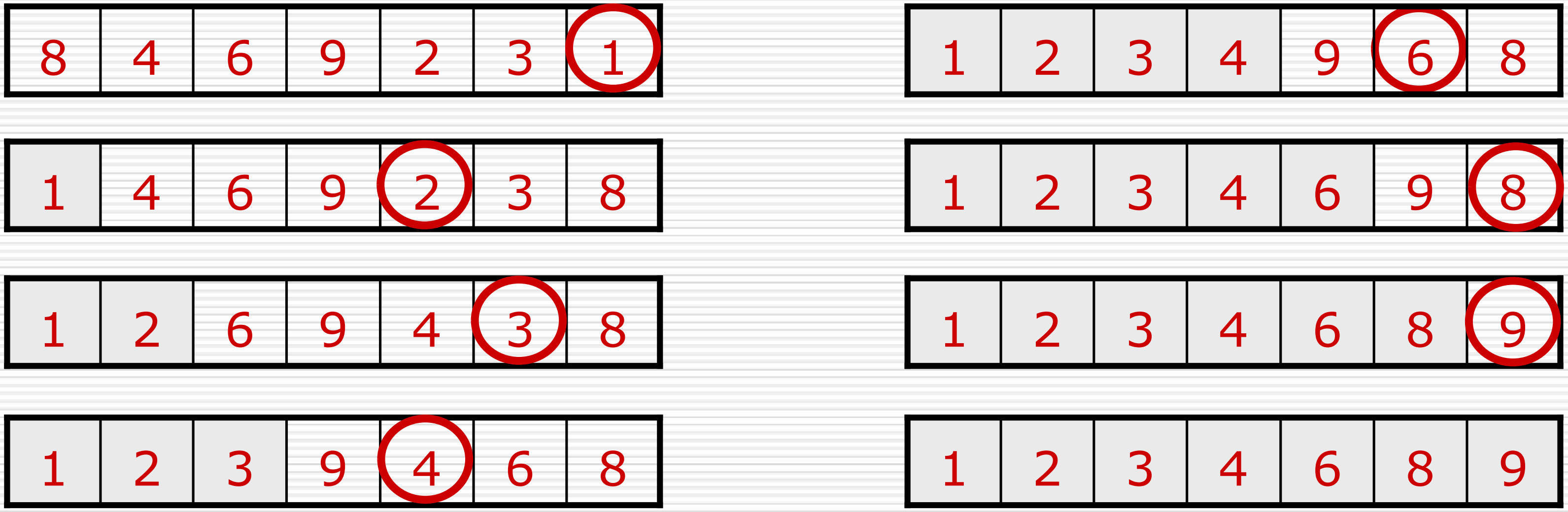
```
// W A P to sort an array with 10 elements using Bubble Sort (Descending)
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,n[10],temp=0;
        clrscr();
        printf("Enter 10 element :");
        for(i=0;i<=9;i++)
                scanf("%d",&n[i]);
        clrscr();
        printf("your array is :\n");
        for(i=0;i<=9;i++)
                printf("%d\t",n[i]);
        for(i=0;i<=9;i++)
        {
                for(j=0;j<=9;j++)
                {
                                if(n[j]<n[j+1])
                                {
                                                temp=n[j];
                                                n[j]=n[j+1];
                                                n[j+1]=temp;
                                }
                }
        }
        printf("Sorted array is :\n");
        for(i=0;i<=9;i++)
                printf("%d\t",n[i]);

        getch();
}
```

Selection Sort

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

How Selection Sort works?



8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

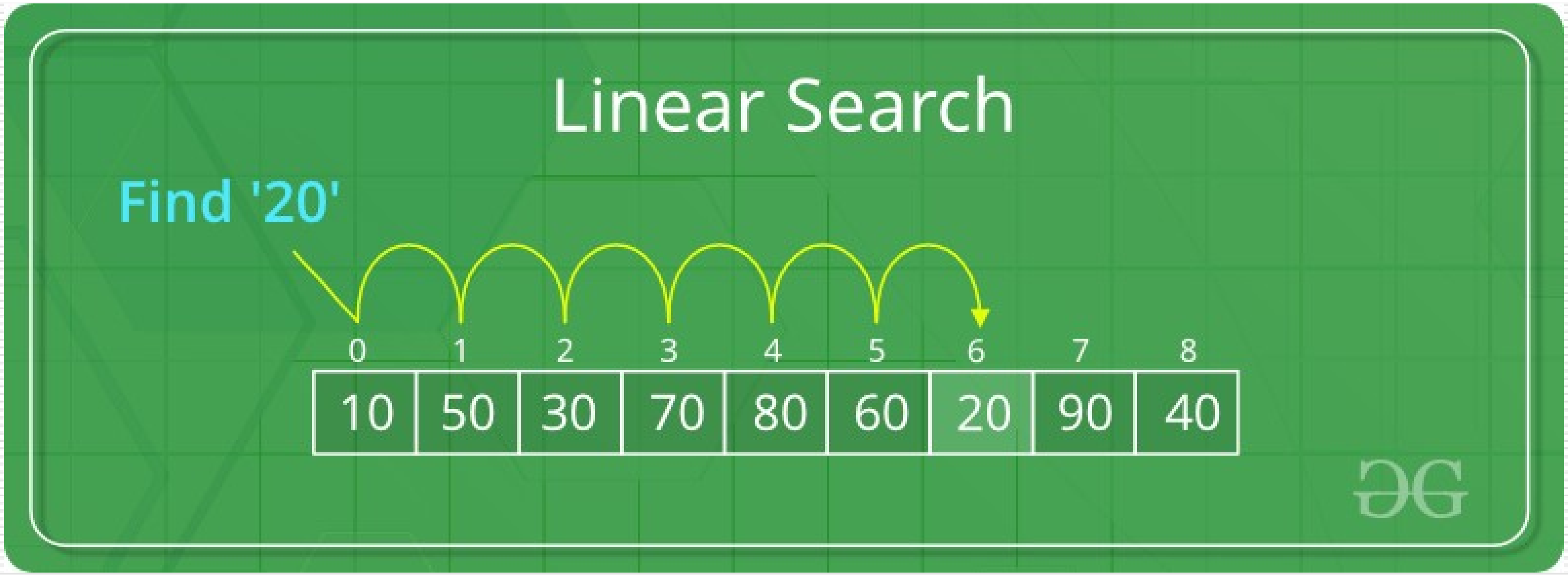
1	2	3	4	6	8	9

```
//Selection Sort
#include<stdio.h>
#include<conio.h>
void main()
{
    int n[10],i,j,p,temp=0;
    clrscr();
    printf("Enter values in array:");
    for(i=0;i<=9;i++)
        scanf("%d",&n[i]);
    clrscr();
    printf("Your Array is:\n");
    for(i=0;i<=9;i++)
        printf("%d\t",n[i]);
    for(i=0;i<=9;i++)
    {
        p=i;
        for(j=i+1;j<=9;j++)
        {
                if(n[p]>n[j])
                        p=j;
        }
        if(p!=j)
        {
                temp=n[i];
                n[i]=n[p];
                n[p]=temp;
        }
    }
    printf("\nAfter Sorting is:\n");
    for(i=0;i<=9;i++)
        printf("%d\t",n[i]);

    getch();
}
```

5.1.2.2 Searching element from array (Linear Search)

- A **linear search**, also known as a **sequential search**, is a method of finding an element within a list.
- It checks each element of the list sequentially until a match is found or the whole list has been searched.



Linear Search
Find '20'
0
1
2
3
4
5
6
7
8
10
50
30
70
80
60
20
90
40

Example:

```
//Linear Search
#include<stdio.h>
#include<conio.h>
void main()
{
        int n[10],i,s=0,f=0;
        clrscr();
        printf("Enter values in array:");
        for(i=0;i<=9;i++)
                scanf("%d",&n[i]);
        clrscr();
        printf("Your Array is:\n");
        for(i=0;i<=9;i++)
                printf("%d\t",n[i]);
        printf("Enter element you want to search :");
        scanf("%d",&s);
        for(i=0;i<=9;i++)
        {
                f=0;
                if(n[i]==s)
                {
                                printf("\nElement is at n[%d] position",i);
                                break;
                }
                else
                {
                                f=1;
                }
        }
        if(f==1)
                printf("\nElement Not Found");
        getch();
}
```

5.1.3 Character Single dimensional Array

5.1.3.1 Character Single dimensional array operations:

5.1.3.2 Use of \0, \n and \t

- A string is actually one-**dimensional array** of **characters** in **C** language. These are often used to create meaningful and readable programs.
- For example: The string "hello world" contains 12 **characters** including '\0' **character** which is automatically added by the compiler at the end of the string.

Declaration

- Declaring a string is as simple as declaring a one-dimensional array. Below is the basic syntax for declaring a string.
  - char str_name[size];

Initializing a String

1. char str[] = “FYBCA";
2. char str[50] = “FYBCA";
3. char str[] = {‘F',‘Y',‘B',‘C',‘A‘,'\0'};
4. char str[6] = {‘F',‘Y',‘B',‘C',‘A','\0'};

Read or Inputting String

5.2 Pointers:

5.2.1 Concepts of Pointers

- The **Pointer in C**, is a variable that stores address of another variable. A **pointer** can also be used to refer to another **pointer** function. A **pointer** can be incremented/decremented, i.e., to point to the next/ previous memory location. The purpose of **pointer** is to save memory space and achieve faster execution time

VARIABLE	POINTER
A **value** stored in a **named** storage/memory address	A **variable** that **points to** the storage/memory address of **another** variable

- every variable has a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

5.2.2 Declaring and initializing int, float, char and void pointers

- The general form of a pointer variable declaration is
  - type *var-name;
- int *ip; // pointer to integer variable
- float *fp; // pointer to float variable
- double *dp; // pointer to double variable
- char *cp; // pointer to char variable

Example

```
void main()
{
    int  var = 20;   /* actual variable declaration */
    int  *ip;        /* pointer variable declaration */

    ip = &var;  /* store address of var in pointer variable*/

    printf("Address of var variable: %u\n", &var  );
    getch();
}
```

Pointer of Pointer

- When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice.
- as is shown below in the example
  - int **var;

Example:

```
void main () {

   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   getch();
}
```

OUTPUT :

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

5.2.3 Pointer to single dimensional numeric array

```
void main()
{
    int my_arr[5] = {1, 2, 3, 4, 5}, i;

    for(i = 0; i < 5; i++)
    {
        printf("Value of a[%d] = %d\t", i, my_arr[i]);
        printf("Address of a[%d] = %u\n", i, &my_arr[i]);
    }

    // signal to operating system program ran fine
    getch();
}
```