

UNIT – IV: Sorting and Searching


Sorting:

Sorting is the process of arranging elements either in ascending (or) descending order


Sorting by selection

- Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.


Working of Selection Sort



Presidency College
(Autonomous)



Reaccredited by
NAAC with A+



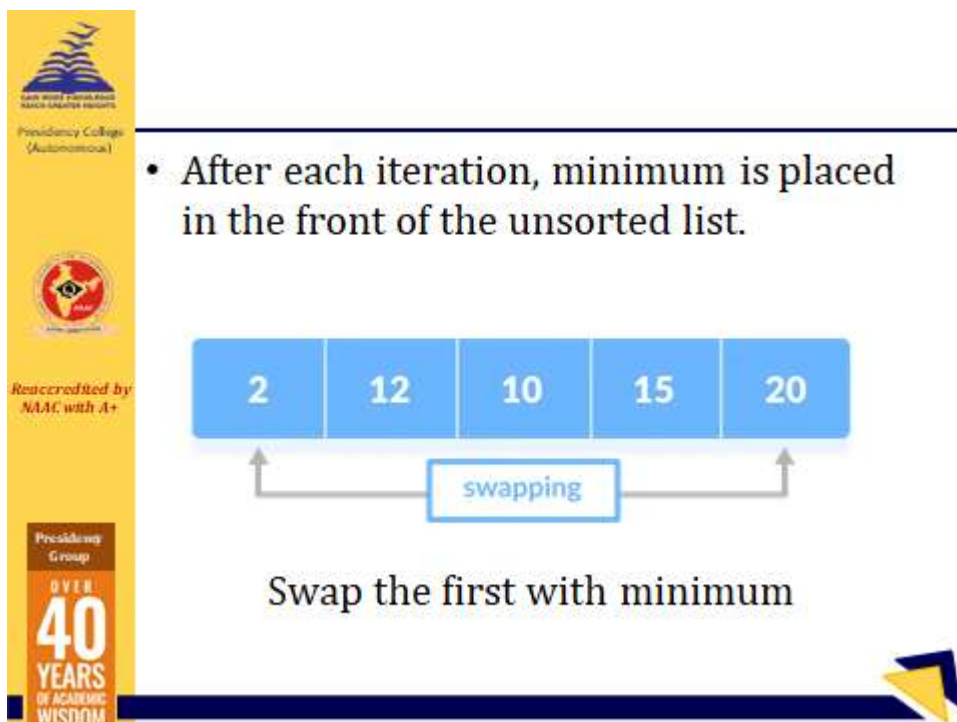
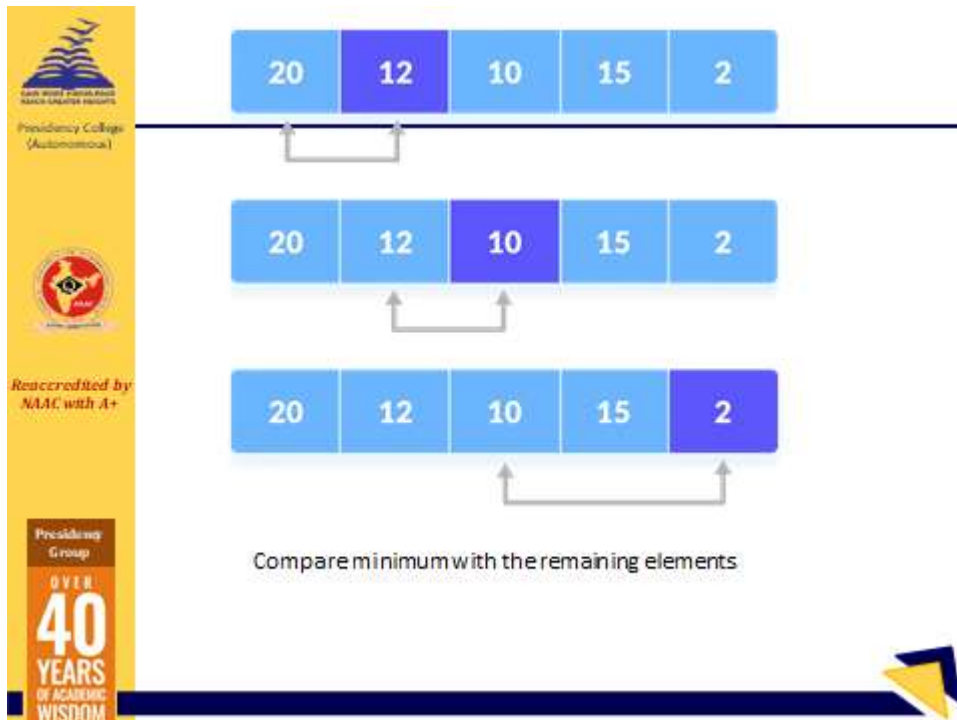
Presidency
Group
OVER
40
YEARS
OF ACADEMIC
WISDOM

Working of Selection Sort

1. Set the first element as minimum

20	12	10	15	2
----	----	----	----	---

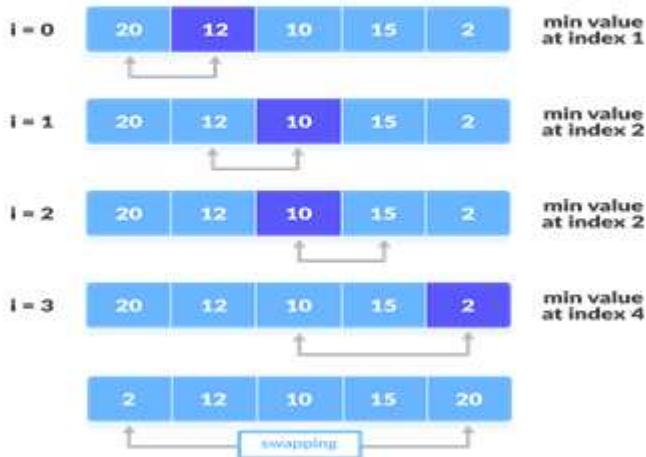
Select first element as minimum
2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.



- For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

The first iteration

step = 0



OVER
40
YEARS
OF ACADEMIC
WISDOM

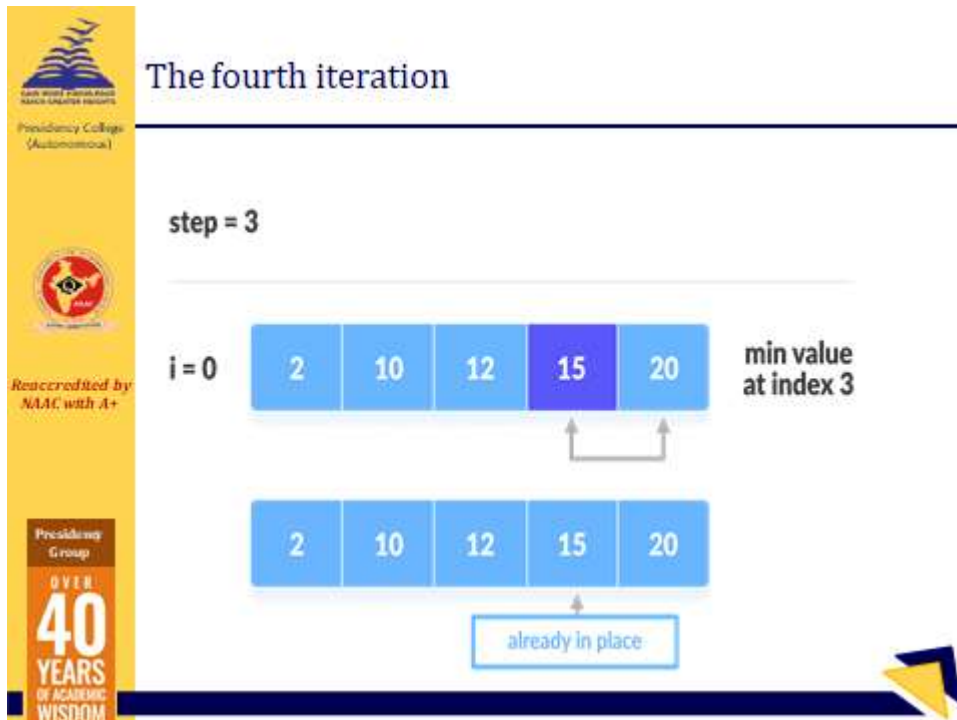
swapping

Presidency
Group

OVER
40
YEARS
OF ACADEMIC
WISDOM

i = 2	2	10	12	15	20	min value at index 2
-------	---	----	----	----	----	-------------------------

already in place




Selection sort Program Example:

```
#include <stdio.h>
void selection(int arr[], int n)
{
    int i, j, small, temp;
    for (i = 0; i < n-1; i++) // One by one move boundary of unsorted subarray
    {
        small = i; //minimum element in unsorted array
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[small])
                small = j;
        // Swap the minimum element with the first element
        temp = arr[small];
        arr[small] = arr[i];
        arr[i] = temp;
    }
}
void printArr(int a[], int n) /* function to print the array */
{
    int i;
    for (i = 0; i < n; i++)
```




```
        printf("%d ", a[i]);  
    }  
    void main()  
    {  
        int a[] = { 12, 31, 25, 8, 32, 17 };  
        int n;  
        clrscr();  
        n = sizeof(a) / sizeof(a[0]);  
        printf("Before sorting array elements are - \n");  
        printArr(a, n);  
        selection(a, n);  
        printf("\nAfter sorting array elements are - \n");  
        printArr(a, n);  
        getch();  
    }
```


sorting by exchange:



Presidency College
(Autonomous)



Reaccredited by
NAAC with A+

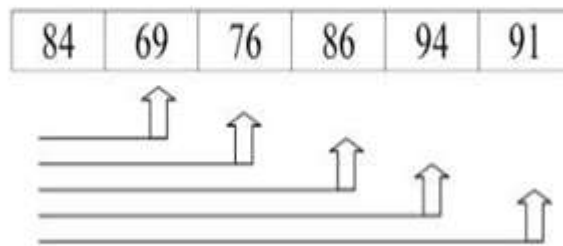


Presidency
Group
OVER
40
YEARS
OF ACADEMIC
WISDOM


Exchange Sort

- The exchange sort is almost similar as the bubble sort
- The exchange sort compares each element of an array and swap those elements that are not in their proper position, just like a bubble sort does. The only difference between the two sorting algorithms is the manner in which they compare the elements.


- The exchange sort compares the first element with each element of the array, making a swap where is necessary.




- When the first pass through the array is complete, the exchange sort then takes the second element and compares it with each following element of the array swapping elements that are out of order. This sorting process continues until the entire array is ordered.



Presidency College
(Autonomous)




Reaccredited by
NAAC with A+



OVER
40
YEARS
OF ACADEMIC
WISDOM


Exchange Sort Working

- List A = {8, 4, 2, 1}
- Exchange Sort finds the smallest value and then the next smallest etc. It does it by pair-wise comparisons and swaps. Element: 1
 - 8 and 4, 4 is smaller so it gets swapped, A = 4, 8, 2, 1
 - 4 and 2, 2 is smaller so it gets swapped, A = 2, 8, 4, 1
 - 2 and 1, 1 is smaller so it gets swapped, A = 1, 8, 4, 2
- The first pass has been completed so now we start the same process starting with the second element in the list. Element: 2
 - 8 and 4, 4 is smaller, swap and A = 1, 4, 8, 2
 - 4 and 2, 2 is smaller, swap and A = 1, 2, 8, 4
- That completes that pass and now the first 2 values are in the correct position. Now the third pass looks at 8 and 4. Element: 3
 - 4 is smaller, swap and A = 1, 2, 4, 8
 - We're done, A = 1, 2, 4, 8




KENNESAW STATE
UNIVERSITY


Insertion Sort:



Presidency College
(Autonomous)



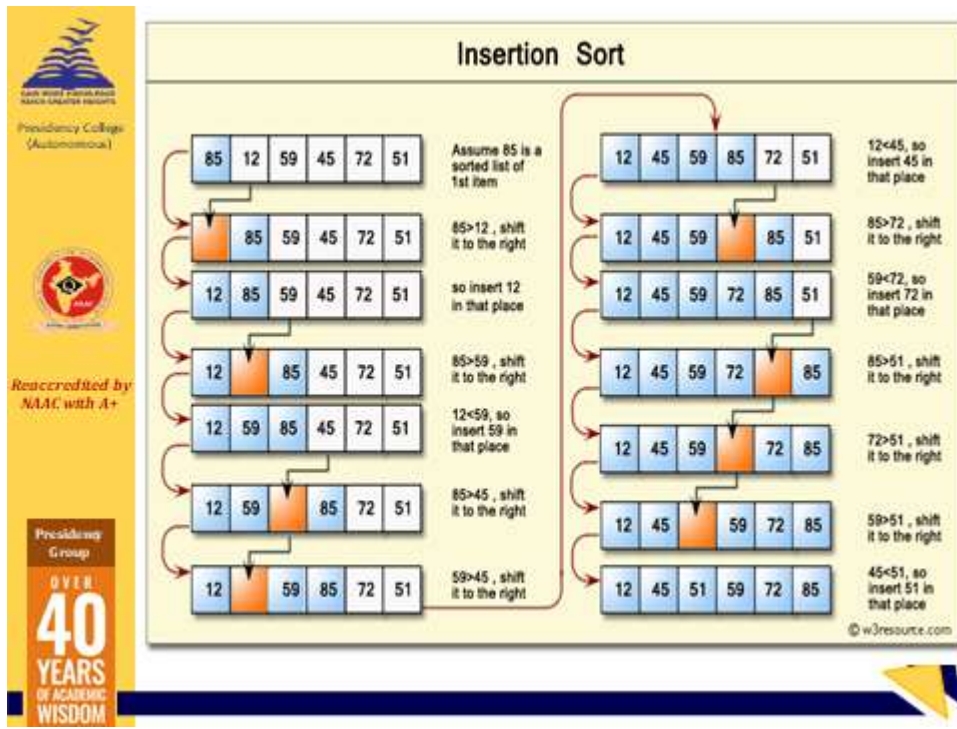
Reaccredited by
NAAC with A+



OVER
40
YEARS
OF ACADEMIC
WISDOM

Insertion sort

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
- The array is virtually split into a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part.



Example:

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

```
void insert(int a[], int n) /* function to sort an array with insertion sort */
```

```
{
```

```
    int i, j, temp;
```

```
    for (i = 1; i < n; i++) {
```

```
        temp = a[i];
```

```
        j = i - 1;
```

```
while(j>=0 && temp <= a[j]) /* Move the elements  
greater than temp to one position ahead from their  
current position*/
```

```
{  
    a[j+1] = a[j];  
    j = j-1;  
}  
a[j+1] = temp;  
}  
}  
void printArr(int a[], int n) /* function to print the array  
*/  
{  
    int i;  
    for (i = 0; i < n; i++)  
        printf("%d ", a[i]);  
}
```

```
void main()  
{  
    int a[] = { 12, 31, 25, 8, 32, 17 };  
    int n = sizeof(a) / sizeof(a[0]);  
    clrscr();  
    printf("Before sorting array elements are - \n");  
    printArr(a, n);  
    insert(a, n);  
    printf("\nAfter sorting array elements are - \n");
```

```
printArr(a, n);
getch();
}
```

Binary Search

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half

steps to perform Binary Search are:


- Begin with an interval covering the whole array.
- If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise, narrow it to the upper half.
- Repeatedly check until the value is found or the interval is empty.

Binary Search


	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
23 > 16 take 2 nd half	L=0	1	2	3	M=4	5	6	7	8	H=9
	2	5	8	12	16	23	38	56	72	91
23 > 56 take 1 st half	0	1	2	3	4	L=5	6	M=7	8	H=9
	2	5	8	12	16	23	38	56	72	91
Found 23, Return 5	0	1	2	3	4	L=5, M=5	H=6	7	8	9
	2	5	8	12	16	23	38	56	72	91

Refer lab program for example of binary search


Text processing and Pattern searching:



Presidency College
(Autonomous)




Reaccredited by
NAAC with A+




Presidency
Group
OVER
40
YEARS
OF ACADEMIC
WISDOM

Pattern searching


- Pattern searching is a very crucial problem in computer science. Whenever we seek for a string in notepad/word file or browser or database or in some information, pattern searching algorithms are used to show the search results.



Presidency College
(Autonomous)



Reaccredited by
NAAC with A+



Presidency
Group
OVER
40
YEARS
OF ACADEMIC
WISDOM

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

int main () {
    char txt[] =
        "tutorialsPointisthebestplatformforprogrammers";
    char pat[] = "a";
    int M = strlen (pat);
    int N = strlen (txt);
    for (int i = 0; i <= N - M; i++) {
        int j;
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;
        if (j == M)
            printf ("Pattern matches at index %d \n", i);
    }
    return 0; }
```




Reaccredited by
NAAC with A+



• Output

Pattern matches at 6

Pattern matches at 25

Pattern matches at 39

Structures and unions:




Reaccredited by
NAAC with A+




structure

- A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type(heterogeneous type of data).
- Each element of a structure is called a member.


Syntax:



Presidency College
(Autonomous)



Reaccredited by
NAAC, with A+




Presidency
Group

OVER
40
YEARS
OF ACADEMIC
WISDOM


- The **,struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

```
struct structure_name  
{  
    data_type member1;  
    data_type member2;  
    .  
    .  
    data_type memeberN;  
};
```


Example:



Presidency College
(Autonomous)



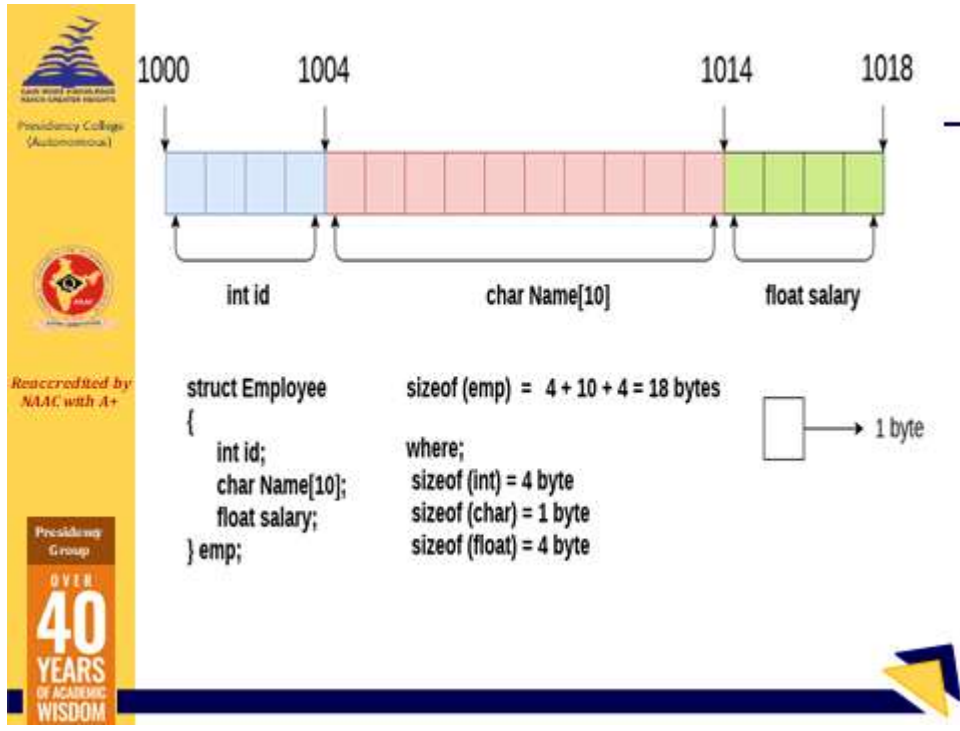
Reaccredited by
NAAC, with A+



Presidency
Group

OVER
40
YEARS
OF ACADEMIC
WISDOM

```
struct employee  
{ int id;  
  char name[20];  
  float salary;  
};
```



- Here, struct is the keyword; employee is the name of the structure; id, name, and salary are the members or fields of the structure.

struct keyword tag or structure tag

```
struct employee{  
  int id;  
  char name[50];  
  float salary;  
};
```

members or
fields of
structure

JavaTpoint.com

Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

- By struct keyword within main() function
- By declaring a variable at the time of defining the structure.



PRESIDENCY COLLEGE

(AUTONOMOUS)

AFFILIATED TO BENGALURU CITY UNIVERSITY, APPROVED BY AICTE, DELHI & RECOGNISED BY THE GOVT. OF KARNATAKA
RE-ACCREDITED BY NAAC WITH 'A++' GRADE



Reaccredited by
NAAC with A+



struct employee

{ **int** id;

char name[50];

float salary;

};

- Now write given code inside the main() function.

struct employee e1, e2;



Reaccredited by
NAAC with A+



2nd way:

struct employee

{ **int** id;

char name[50];

float salary;

}e1,e2;



Accessing members of the structure

There are two ways to access structure members:

- By . (member or dot operator)
- By -> (structure pointer operator)

```
#include<stdio.h>
#include <string.h>
struct employee
```

```
{ int id;
  char name[50];
}e1; //declaring e1 variable for structure
```

```
int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char
  array
  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
  return 0;
}
```




PRESIDENCY COLLEGE (AUTONOMOUS)

AFFILIATED TO BENGALURU CITY UNIVERSITY, APPROVED BY AICTE, DELHI & RECOGNISED BY THE GOVT. OF KARNATAKA
RE-ACCREDITED BY NAAC WITH 'A++' GRADE



output

employee 1 id : 101

employee 1 name : Sonoo Jaiswal



Reaccredited by
NAAC with A+



Union:

A union is a user-defined type similar to [structs in C](#) except for one key difference.

Structures allocate enough space to store all their members, whereas **unions** can only hold one member value at a time.



PRESIDENCY COLLEGE

(AUTONOMOUS)

AFFILIATED TO BENGALURU CITY UNIVERSITY, APPROVED BY AICTE, DELHI & RECOGNISED BY THE GOVT. OF KARNATAKA
RE-ACCREDITED BY NAAC WITH 'A++' GRADE



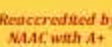
Union

- Union is an user defined datatype in C programming language.
- It is a collection of variables of different datatypes in the same memory location.
- We can define a union with many members, but at a given point of time only one member can contain a value



Need for Union in C programming

- C unions are used to save memory.
- C unions allow data members which are mutually exclusive to share the same memory



```
union union_name
{
    datatype field_name;
    datatype field_name;
    // more variables
};
```

Note : Size of the union is the the size of its largest field because sufficient number of bytes must be reserved to store the largest sized field.

- To access the fields of a union, use dot(.) operator i.e., the variable name followed by dot operator followed by field name.

Example:



```
#include <stdio.h>
union Job {
    float salary;
    int workerNo;
} j;

int main() {
    j.salary = 12.3;

    // when j.workerNo is assigned a value,
    // j.salary will no longer hold 12.3
    j.workerNo = 100;

    printf("Salary = %.1f\n", j.salary);
    printf("Number of workers = %d", j.workerNo);
    return 0;
}
```

Output

```
Salary = 0.0
Number of workers = 100
```

Differences between Structure and Union:



	STRUCTURE	UNION
Keyword	The keyword struct is used to define a structure	The keyword union is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members.	When a variable is associated with a union, the compiler allocates the memory by considering the size of the largest member. So, size of union is equal to the size of largest member.
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Difference between unions and structures

Let's take an example to demonstrate the difference between unions and structures:

```
#include <stdio.h>
union unionJob
{
    //defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;

struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;
```



```
int main()
{
    printf("size of union = %d bytes", sizeof(uJob));
    printf("\nsize of structure = %d bytes", sizeof(sJob));
    return 0;
}
```

Output

```
size of union = 32
size of structure = 40
```

Command line arguments.



command line arguments

- It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

```
int main(int argc, char *argv[] )
```

- The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program

Example:

```
#include <stdio.h>

Void main( int argc, char *argv[] ) {
    if( argc == 2 ) {
        printf("The argument supplied is %s\n",
            argv[1]); }
    else if( argc > 2 ) {
        printf("Too many arguments supplied.\n");
    }
    else {
        printf("One argument expected.\n");
    }
}
```