Unit-4

Structures and unions: using structures and unions, comparison of structure with arrays and union.

Pointers: pointer data type, pointer declaration, initialization, accessing values using pointers, pointers and arrays.

File handling: Definition of Files, Opening modes of files, Standard function-(fopen ( ), fclose ( ), feof ( ), fseek ( ), rewind ( )).

Why use structure?

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity **Student** may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

•Construct individual arrays for storing names, roll numbers, and marks.

•Use a special data structure to store the collection of different data types.

What is Structure

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information

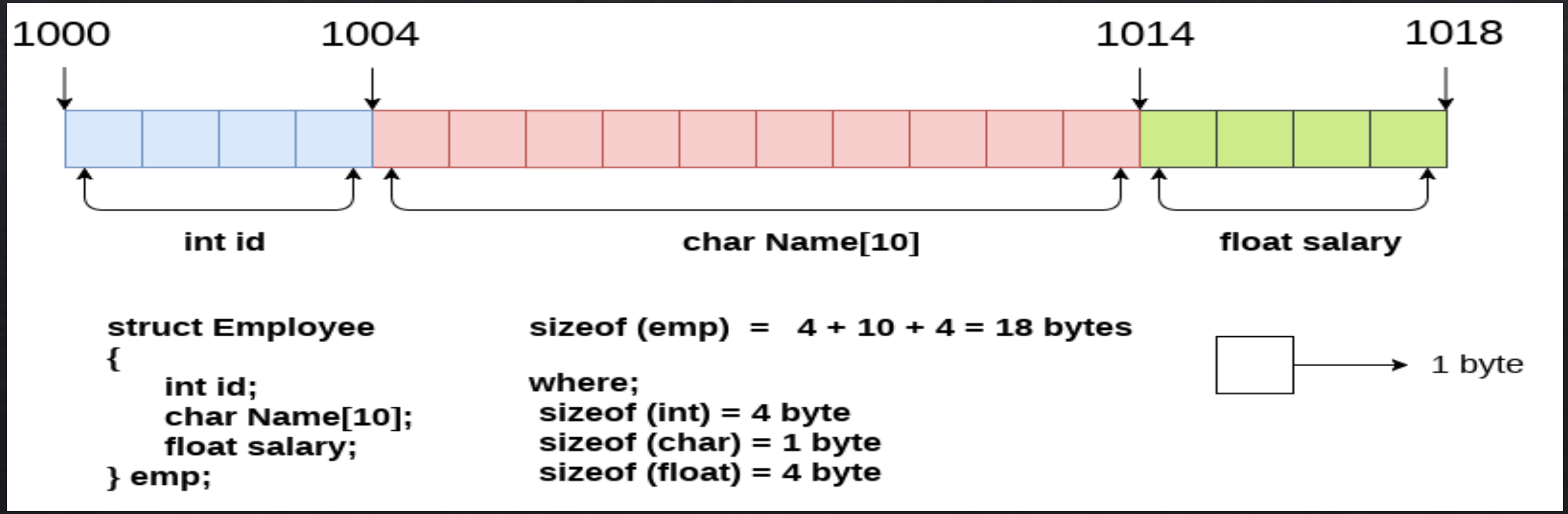
The **,struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

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

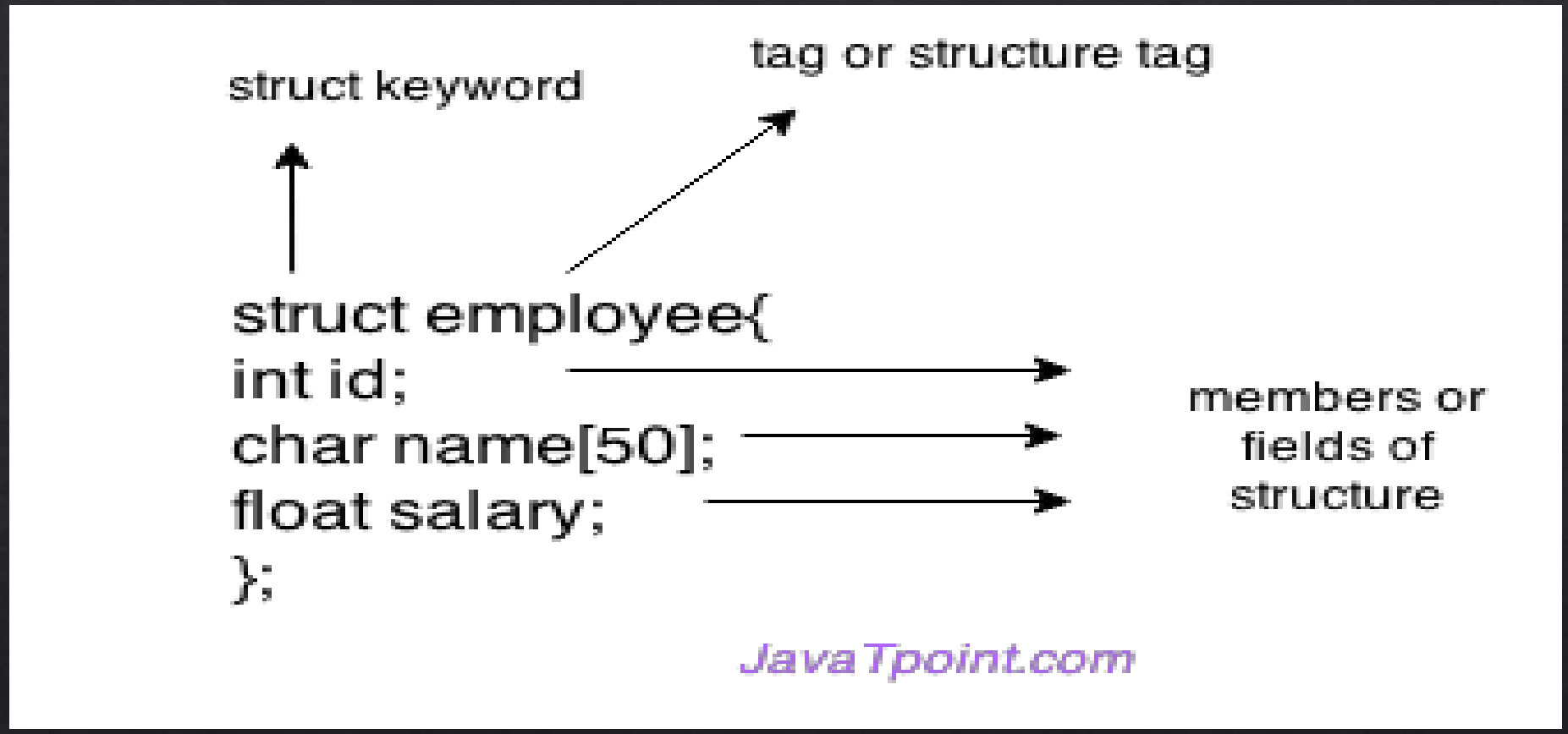
Let's see the example to define a structure for an entity employee in c.

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

The following image shows the memory allocation of the structure employee that is defined in the above example.



Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



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.

1st way:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

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

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way:

Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;
```

Accessing members of the structure

There are two ways to access structure members:

1.By . (member or dot operator)

2.By -> (structure pointer operator)

Let's see the code to access the *id* member of *p1* variable by . (member) operator.

C Structure example

Let's see a simple example of structure in C language.

```
#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, “David");//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;
}
```

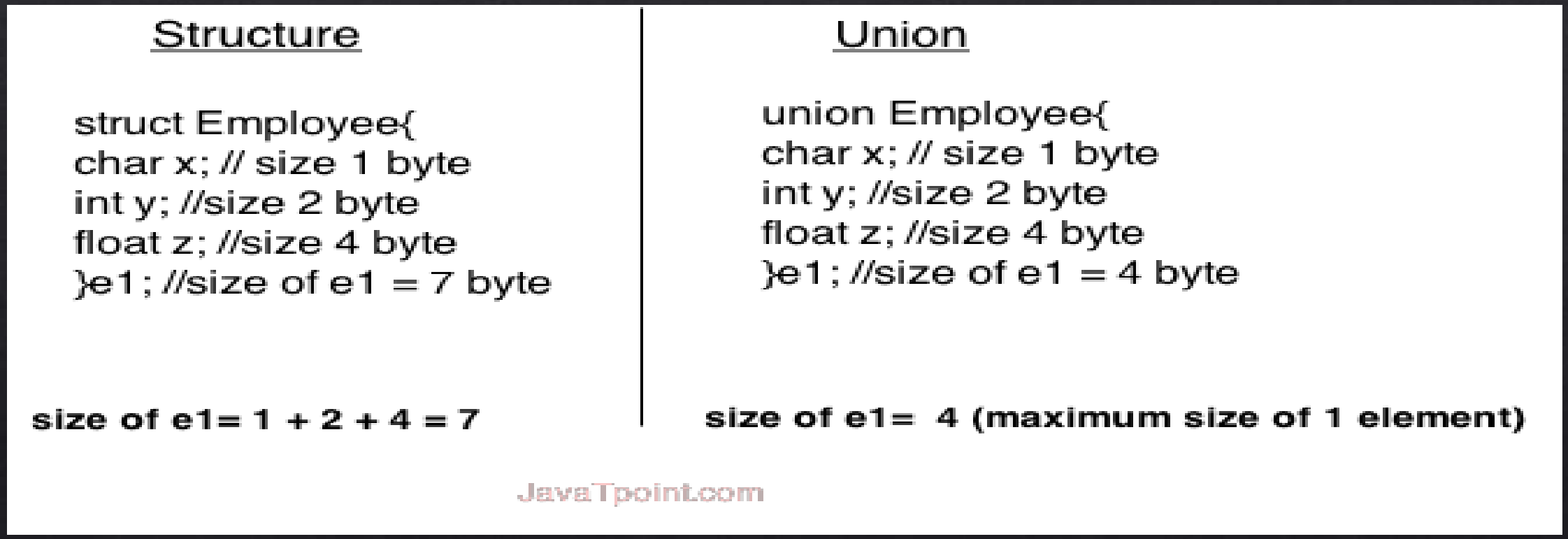
Output:

```
employee 1 id : 101
employee 1 name : David
```

C Union

Like structure, **Union in c language** is *a user-defined data type* that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.



Advantage of union over structure

It **occupies less memory** because it occupies the size of the largest member only.

Disadvantage of union over structure

Only the last entered data can be stored in the union. It overwrites the data previously stored in the union.

Defining union

The **union** keyword is used to define the union. Let's see the syntax to define union in c.

```
union union_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Let's see the example to define union for an employee in c.

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

C Union example

Let's see a simple example of union in C language.

```
#include <stdio.h>
#include <string.h>
union employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for union

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;
}
```

Output:

```
employee 1 id : 1869508435
employee 1 name : Sonoo Jaiswal
```

Comparison of Structure with 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 memory. 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.

Comparison of Structure with Arrays

Arrays	Structures
1. An array is a collection of related data elements of the same type.	1. Structure can have elements of different types
2. An array is a derived data type	2. A structure is a programmer-defined data type
3. Any array behaves like a built-in data types. All we have to do is to declare an array variable and use it.	3. But in the case of structure, first, we have to design and declare a data structure before the variable of that type are declared and used.
4. Array allocates static memory and uses index/subscript for accessing elements of the array.	4. Structures allocate dynamic memory and uses (.) operator for accessing the member of a structure.
5. An array is a pointer to the first element of it	5. Structure is not a pointer
6. Element access takes relatively less time.	6. Property access takes relatively large time.

C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;
int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.
```

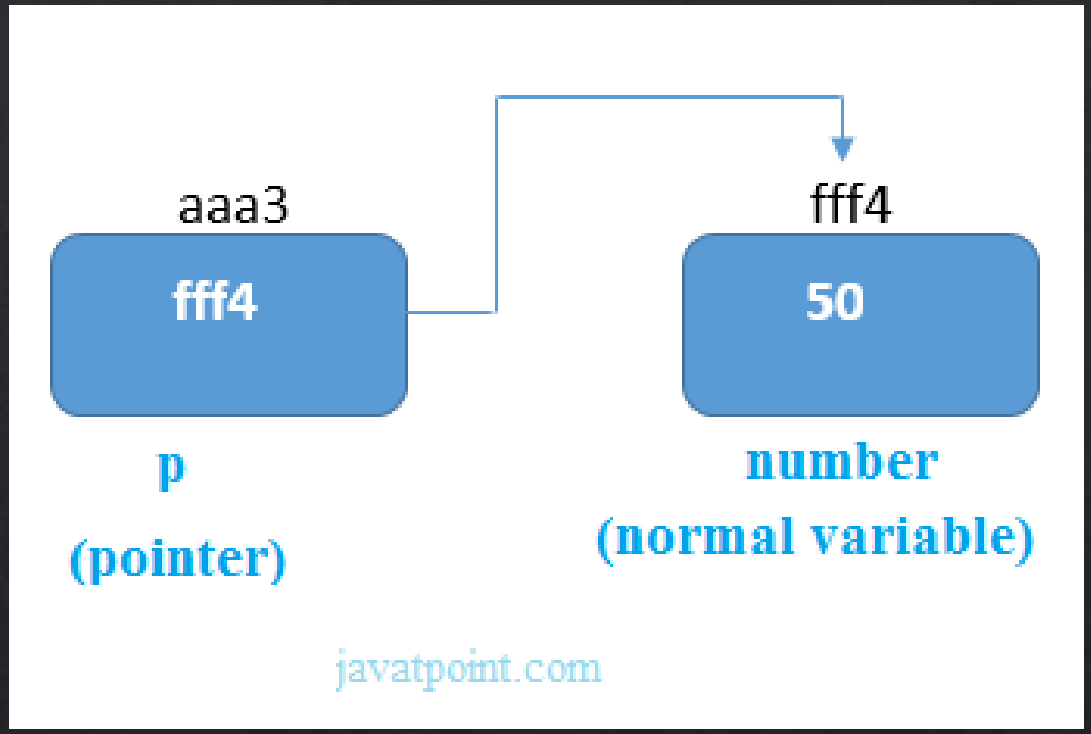
Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
int *a;//pointer to int
char *c;//pointer to char
```

Pointer Example

An example of using pointers to print the address and value is given below.



As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

```
#include<stdio.h>
int main(){
int number=50;
int *p;
p=&number;//stores the address of number variable

printf("Address of number variable is %x \n", &number);

printf("Address of p variable is %x \n",p); // p contains the address of the number therefore printing p gives the address of number.

printf("Value of p variable is %d \n",*p); // As we know that * is used to dereference a pointer therefore if we print *p, we will get the value stored at the address contained by p.

return 0;
}
```

Output

```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

Advantage of pointer

1) Pointer **reduces the code** and **improves the performance**, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.

2) We can **return multiple values from a function** using the pointer.

3) It makes you able to **access any memory location** in the computer's memory.

Usage of pointer

There are many applications of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2) Arrays, Functions, and Structures

Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

Pointer to array

```
int arr[10];
int *p[10]=&arr; // Variable p of type pointer is pointing to the address of an integer array arr.
 #include <stdio.h>
  int main( )
 {
 int *p,i;

 int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;

 p = &val[0];

 for ( i = 0 ; i<7 ; i++ )
 {
 printf("val[%d]: value is %d and address is %x\n", i, *p, p);

 p++;

  }
 return 0;
 }
```

Output:

```
val[0]: value is 11 and address is 60feec
val[1]: value is 22 and address is 60fef0
val[2]: value is 33 and address is 60fef4
val[3]: value is 44 and address is 60fef8
val[4]: value is 55 and address is 60fefc
val[5]: value is 66 and address is 60ff00
val[6]: value is 77 and address is 60ff04
```

File Handling in C

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file

How to Create a File

Whenever you want to work with a file, the first step is to create a file. A file is nothing but space in a memory where data is stored.

To create a file in a 'C' program following syntax is used,

FILE *fp; fp = fopen ("file_name", "mode");

In the above syntax, the file is a data structure which is defined in the standard library.

fopen is a standard function which is used to open a file.

•If the file is not present on the system, then it is created and then opened.

•If a file is already present on the system, then it is directly opened using this function.

fp is a file pointer which points to the type file.

Whenever you open or create a file, you have to specify what you are going to do with the file. A file in 'C' programming can be created or opened for reading/writing purposes. A mode is used to specify whether you want to open a file for any of the below-given purposes. Following are the different types of modes in 'C' programming which can be used while working with a file.

Different operations that can be performed on a file are:

1. Creation of a new file (**fopen with attributes as “a” or “a+” or “w” or “w++”**)
2. Opening an existing file (**fopen**)
3. Reading from file (**fscanf or fgetc**)
4. Writing to a file (**fprintf or fputs**)
5. Moving to a specific location in a file (**fseek, rewind**)
6. Closing a file (**fclose**)

File Mode	Description
r	Open a file for reading. If a file is in reading mode, then no data is deleted if a file is already present on a system.
w	Open a file for writing. If a file is in writing mode, then a new file is created if a file doesn't exist at all. If a file is already present on a system, then all the data inside the file is truncated, and it is opened for writing purposes.
a	Open a file in append mode. If a file is in append mode, then the file is opened. The content within the file doesn't change.
r+	open for reading and writing from beginning
w+	open for reading and writing, overwriting a file
a+	open for reading and writing, appending to file

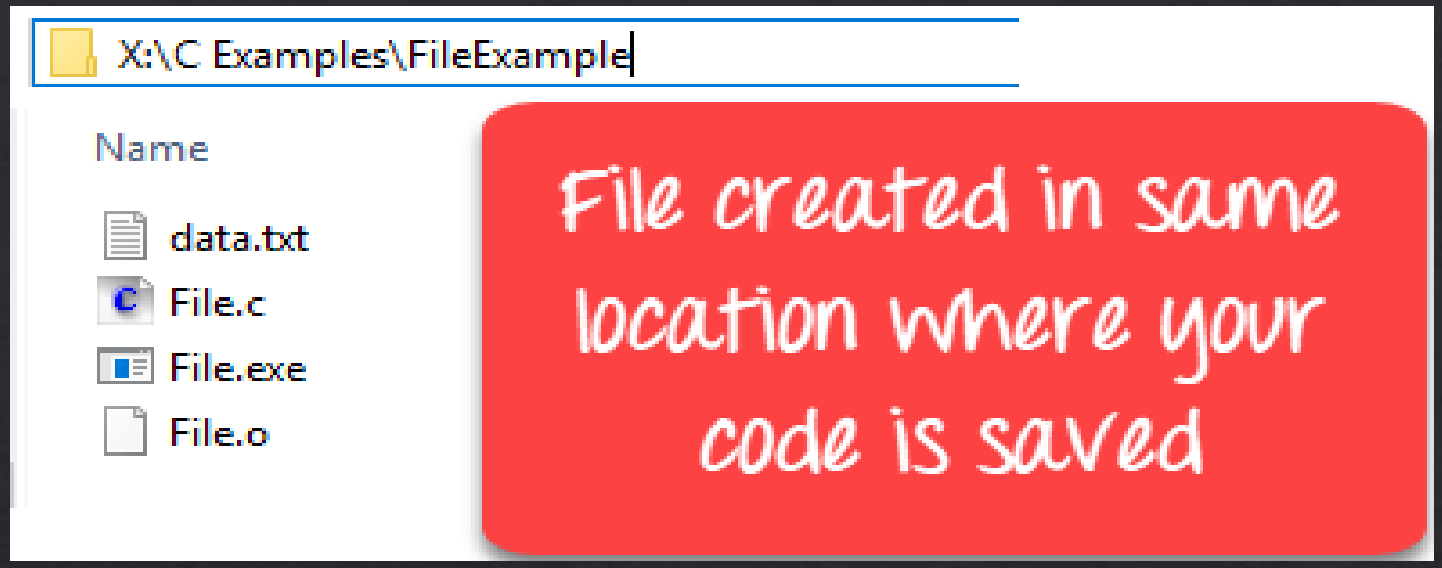
In the given syntax, the filename and the mode are specified as strings hence they must always be enclosed within double quotes.

Example:

```
#include <stdio.h>
 int main()
{ FILE *fp; fp = fopen ("data.txt", "w");
}
```

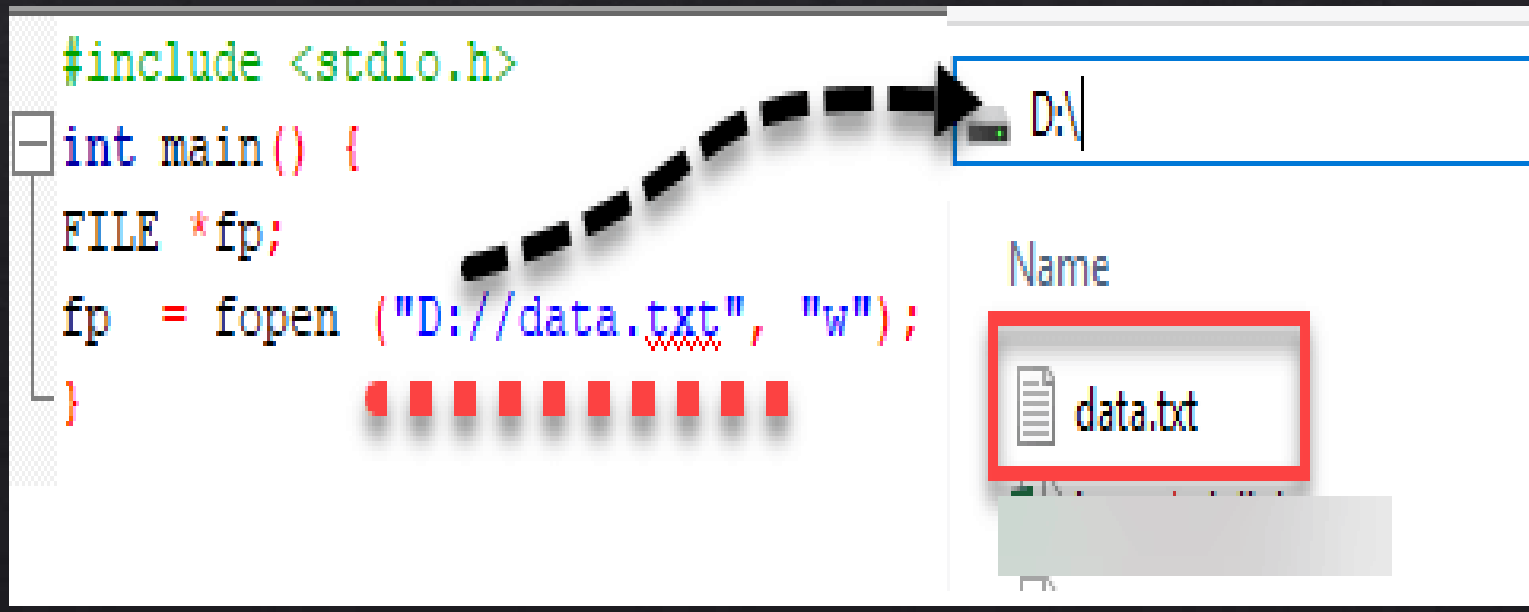
Output:

File is created in the same folder where you have saved your code.



You can specify the path where you want to create your file

```
#include <stdio.h>
 int main()
 {
 FILE *fp; fp = fopen ("D://data.txt", "w");
}
```



How to Close a file

One should always close a file whenever the operations on file are over. It means the contents and links to the file are terminated. This prevents accidental damage to the file.

'C' provides the fclose function to perform file closing operation. The syntax of fclose is as follows,

fclose (file_pointer);

Example:

```
FILE *fp; fp = fopen ("data.txt", "r");
fclose (fp);
```

The fclose function takes a file pointer as an argument. The file associated with the file pointer is then closed with the help of fclose function. It returns 0 if close was successful and EOF (end of file) if there is an error has occurred while file closing.

After closing the file, the same file pointer can also be used with other files.

In 'C' programming, files are automatically close when the program is terminated. Closing a file manually by writing fclose function is a good programming practice.

Writing File : fprintf() function

The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

Syntax:

int fprintf(FILE *stream, const char *format [, argument, ...])

Example:

```
#include <stdio.h>
main(){
   FILE *fp;
   fp = fopen("file.txt", "w");//opening file
   fprintf(fp, "Hello file by fprintf");//writing data into file
   fclose(fp);//closing file
}
```

Reading File : fscanf() function

The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

Syntax:

int fscanf(FILE *stream, const char *format [, argument, ...])

Example:

```
#include <stdio.h>
main(){
   FILE *fp;
   char buff[255];//creating char array to store data of file
   fp = fopen("file.txt", "r");
   while(fscanf(fp, "%s", buff)!=EOF){
   printf("%s ", buff );
   }
   fclose(fp);
}
```

Output:

```
Hello file by fprintf...
```

C File Example: Storing employee information

Let's see a file handling example to store employee information as entered by user from console. We are going to store id, name and salary of the employee.

```
#include <stdio.h>
void main()
{
    FILE *fptr;
    int id;
    char name[30];
    float salary;
    fptr = fopen("emp.txt", "w");/*  open for writing */
    if (fptr == NULL)
    {
        printf("File does not exists \n");
        return;
    }
    printf("Enter the id\n");
    scanf("%d", &id);
    fprintf(fptr, "Id= %d\n", id);
    printf("Enter the name \n");
    scanf("%s", name);
    fprintf(fptr, "Name= %s\n", name);
    printf("Enter the salary\n");
    scanf("%f", &salary);
    fprintf(fptr, "Salary= %f\n", salary);
    fclose(fptr);
}
```

Output:

```
Enter the id 1
Enter the name sonoo
Enter the salary 120000
```

Writing File : fputc() function

The fputc() function is used to write a single character into file. It outputs a character to a stream.

Syntax:

int fputc(int c, FILE *stream)

Example:

```
#include <stdio.h>
main(){
   FILE *fp;
   fp = fopen("file1.txt", "w");//opening file
   fputc('a',fp);//writing single character into file
   fclose(fp);//closing file
}
```

file1.txt

a

Reading File : fgetc() function

The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

Syntax:

int fgetc(FILE *stream)

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("myfile.txt","r");
while((c=fgetc(fp))!=EOF){
printf("%c",c);
}
fclose(fp);
getch();
}
```

myfile.txt

this is simple text message

C fseek() function

The fseek() function is used to set the file pointer to the specified offset. It is used to write data into file at desired location.

Syntax:

int fseek(FILE *stream, long int offset, int whence)

There are 3 constants used in the fseek() function for whence: SEEK_SET, SEEK_CUR and SEEK_END.

Example:

```
#include<stdio.h>
main()
{
    FILE *fp;
    char arr[20];
    fp=fopen("test.txt","w+");

    fprintf(fp,"program is made by sita");
   fseek(fp,19,SEEK_SET);
   fprintf(fp,"Monu");
    fclose(fp);
}
```

test.txt

Program is made by Monu

C rewind() function

The rewind() function sets the file pointer at the beginning of the stream. It is useful if you have to use stream many times.

Syntax:

```
void rewind(FILE *stream)
```

Example:

File: file.txt
this is a simple text
File: rewind.c

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("file.txt","r");
while((c=fgetc(fp))!=EOF)
{
printf("%c",c);
}
rewind(fp);//moves the file pointer at beginning of the file
while((c=fgetc(fp))!=EOF){
printf("%c",c);
}
fclose(fp);
getch();
}
```

Output:
this is a simple text this is a simple text

/*As you can see, rewind() function moves the file pointer at beginning of the file that is why "this is simple text" is printed 2 times. If you don't call rewind() function, "this is simple text" will be printed only once.*/

Now open file from current directory, you will see emp.txt file. It will have following information.

emp.txt

Id= 1
Name= sonoo
Salary= 120000

C feof() function

C feof function is used to determine if the end of the file (stream), specified has been reached or not. This function keeps on searching the end of file (eof) in your file program.

```
#include <stdio.h>
int main()
{
  FILE *fp = fopen("sample.txt", "r");
  int ch = getc(fp);
  while (ch != EOF)
  {
    putchar(ch);
    ch = getc(fp);
  }
  if (feof(fp))
     printf("\nEnd of file reached.");
  else
     printf("\nSomething went wrong.");
  fclose(fp);
  getchar();
  return 0;
}
```

Output:

```
hello world
End of file reached.
```