Department Of Computer Science
B.Sc-I
Semester II
Subject- Programming Skills Using 'C'

Unit-II

Structures & File handling

(a)Structure & Dynamic memory Allocation:

- ## User Defined Data Types:

C allows the feature called type definition which allows programmers to define their identifier that would represent an existing data type. There are three such types:

Data Types	Description
Structure	It is a package of variables of different types under a single name. This is done to handle data efficiently. "struct" keyword is used to define a structure.
Union	These allow storing various data types in the same memory location. Programmers can define a union with different members, but only a single member can contain a value at a given time. It is used for
Enum	Enumeration is a special data type that consists of integral constants, and each of them is assigned with a specific name. "enum" keyword is used to define the enumerated data type.

1) Structure:

The structure is a user-defined data type in C, which is used to store a collection of different kinds of data.

- The structure is something similar to an array; the only difference is array is used to store the same data types.
- *Struct* keyword is used to declare the structure in C.
- Variables inside the structure are called *members of the structure*.

- **Defining Structure:**

Syntax:

```
struct structureName
{
  //member definitions
};
```

Example:

```
struct Courses
{
  char  WebSite[50];
  char  Subject[50];
  int   Price;
  };
```

- **Accessing Structure Member:**

Example:

```
#include<stdio.h>
#include<conio.h>
struct Courses
{
  char  WebSite[50];
  char  Subject[50];
  int   Price;
};
 void main( )
{
  struct Courses C;
  //Initialization
  strcpy( C.WebSite, "w3schools.in");
  strcpy( C.Subject, "The C Programming Language");
  C.Price = 0;
  //Print
  printf( "WebSite : %s\n", C.WebSite);
  printf( "Book Author : %s\n", C.Subject);
  printf( "Book Price : %d\n", C.Price);
}
```

Output:
WebSite : w3schools.in
Book Author: The C Programming Language
Book Price : 0

2) union:

Unions are user-defined data type in C, which is used to store a collection of different kinds of data, just like a structure. However, with unions, you can only store information in one field at any one time.

- Unions are like structures except it used less memory.
- The keyword union is used to declare the structure in C.
- Variables inside the union are called members of the union.

- **Defining union:**

```
union unionName
{
  //member definitions
};
```

Example:

```
union Courses
{
  char  WebSite[50];
  char  Subject[50];
  int   Price;
};
```

- **Accessing union Member:**

```
#include<stdio.h>
#include<conio.h>
 union Courses
{
  char  WebSite[50];
  char  Subject[50];
  int   Price;
};

void main( )
```

```
{
  union Courses C;
  strcpy( C.WebSite, "w3schools.in");
  printf( "WebSite : %s\n", C.WebSite);
  strcpy( C.Subject, "The C Programming Language");
  printf( "Book Author : %s\n", C.Subject);
  C.Price = 0;
  printf( "Book Price : %d\n", C.Price);
}
```

Output:
WebSite : w3schools.in
Book Author: The C Programming Language
Book Price : 0

3)typedef:

typedef is a C keyword implemented to tell the compiler for assigning an alternative name to C's already exist data types.
This keyword, typedef typically employed in association with user-defined data types in cases if the names of datatypes turn out to be a little complicated or intricate for a programmer to get or to use within programs.
The typical format for implementing this typedef keyword is:

Syntax:

typedef <existing_names_of_datatype> <alias__userGiven_name>;

- ## Nested Structure:

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.

The structure variables can be a normal structure variable or a pointer variable to access the data.

Example:

```
#include<stdio.h>
struct address
{
   char city[20];
   int pin;
   char phone[14];
};
struct employee
{
   char name[20];
   struct address add;
};
void main ()
{
   struct employee emp;
   printf("Enter employee information?\n");
   scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin,
emp.add.phone);
   printf("Printing the employee information....\n");
   printf("name: %s\nCity: %s\nPincode: %d\nPhone:
%s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
}
```

Output:

Enter employee information?

Amar

Dahiwadi

415508

9099868686

Printing the employee information....

name: Amar

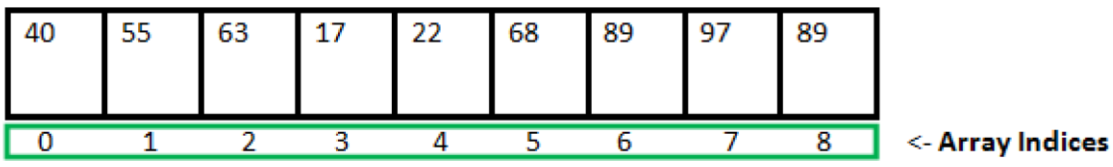
City: Dahiwadi

Pincode: 415508

Phone: 9099868686

- ## Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc():

Since C is a structured language, it has some fixed rules for programming. One of it includes changing the size of an array. An array is collection of items stored at continuous memory locations.



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

As it can be seen that the length (size) of the array above made is 9. But what if there is a requirement to change this length (size). For Example,

- If there is a situation where only 5 elements are needed to be entered in this array. In this case, the remaining 4 indices are just wasting memory in this

array. So there is a requirement to lessen the length (size) of the array from 9 to 5.

- Take another situation. In this, there is an array of 9 elements with all 9 indices filled. But there is a need to enter 3 more elements in this array. In this case 3 indices more are required. So the length (size) of the array needs to be changed from 9 to 12.

This procedure is referred to as **Dynamic Memory Allocation in C**.

C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming. They are:

1. malloc()
2. calloc()
3. free()
4. realloc()

Let's look at each of them in greater detail.

1) malloc() method :

"malloc" or "memory allocation" method in C is used to dynamically allocate a single large block of memory with the specified size.
It returns a pointer of type void which can be cast into a pointer of any form.

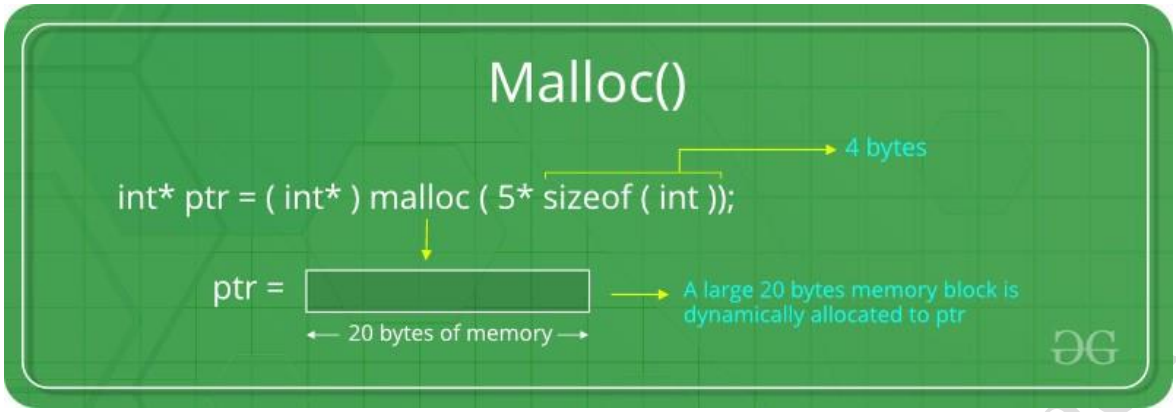
Syntax:

ptr = (cast-type*) malloc(byte-size)

For Example:

ptr = (int*) malloc(100 * sizeof(int));

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.



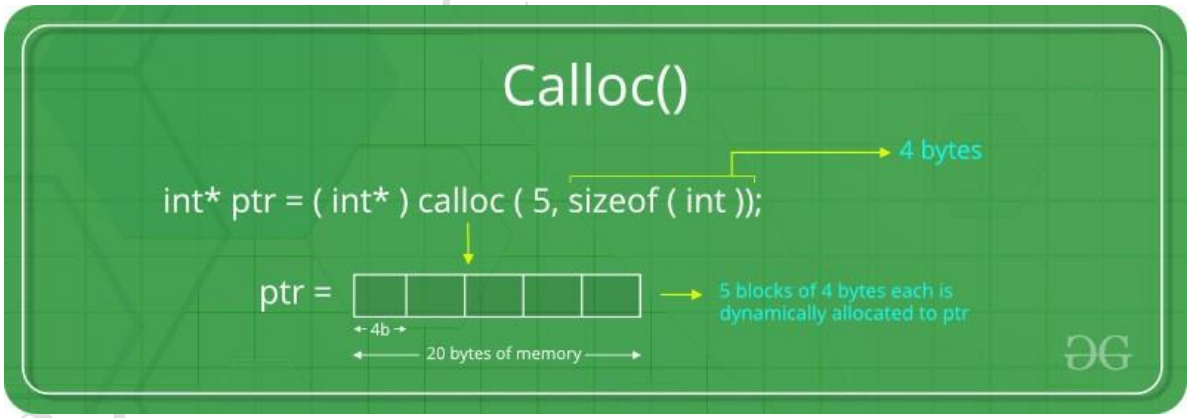
If space is insufficient, allocation fails and returns a NULL pointer.

2) calloc() method

"calloc" or "contiguous allocation" method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value '0'.

Syntax:
ptr = (cast-type*)calloc(n, element-size);

For Example:
ptr = (float*) calloc(25, sizeof(float));
This statement allocates contiguous space in memory for 25 elements each with the size of the float.



If space is insufficient, allocation fails and returns a NULL pointer.

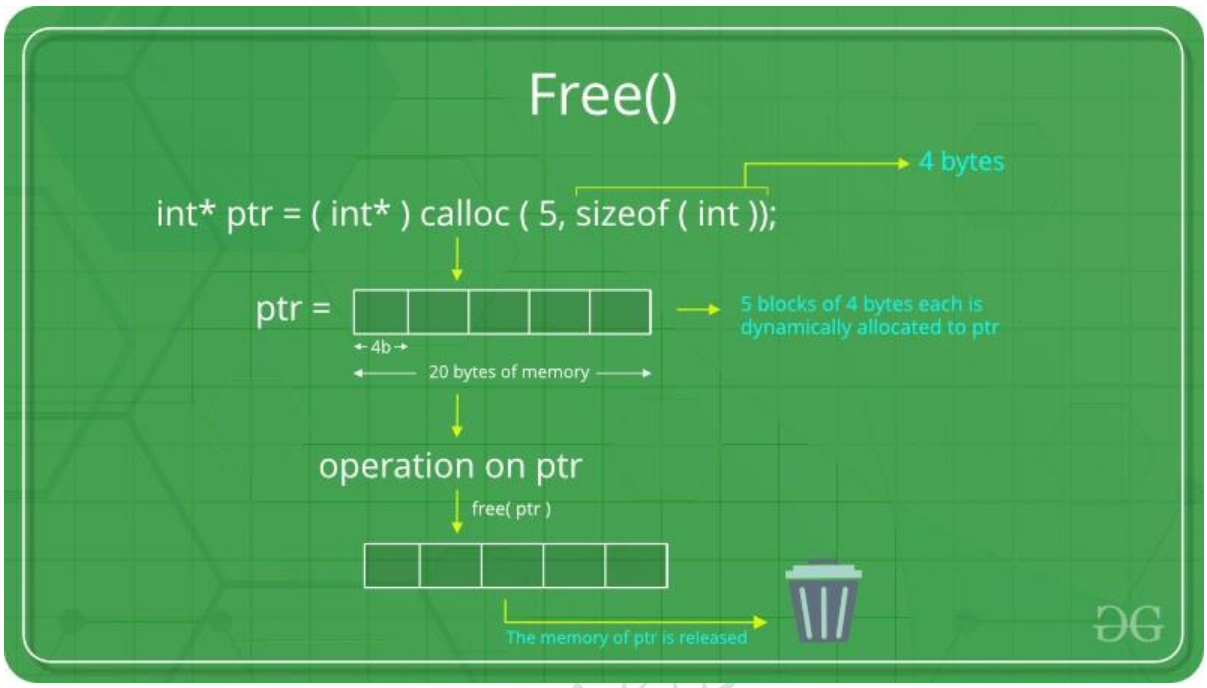
3. free() method

"free" method in C is used to dynamically de-allocate the memory.
The memory allocated using functions malloc() and calloc() is not de-allocated on their own.
Hence the free() method is used, whenever the dynamic memory allocation takes place.

It helps to reduce wastage of memory by freeing it.

Syntax:

free(ptr);



4. realloc() method

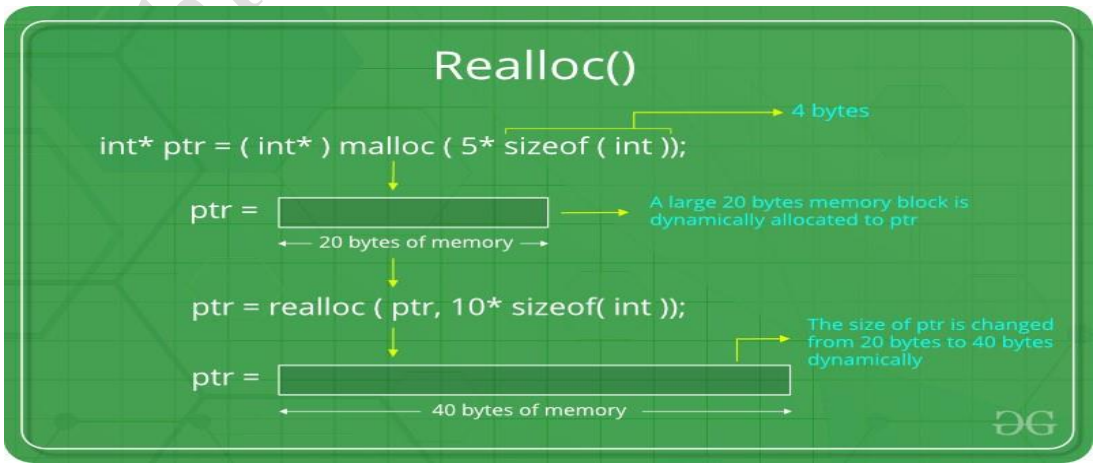
"realloc" or "re-allocation" method in C is used to dynamically change the memory allocation of a previously allocated memory.

In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory.

Syntax:

ptr = realloc(ptr, newSize);

where ptr is reallocated with new size 'newSize'.



FILE HANDLING :

In any programming language it is vital to learn file handling techniques. Many applications will at some point involve accessing folders and files on the hard drive. In C, a stream is associated with a file. Special functions have been designed for handling file operations. Some of them will be discussed in this chapter. The header file stdio.h is required for using these functions.

- **Declaring a File Pointer:**

In C language, in order to declare a file, we use a file pointer.

A file pointer is a pointer variable that specifies the next byte to be read or written to. Every time a file is opened, the file pointer points to the beginning of the file.

A file is declared as follows:

FILE *fp;

//fp is the name of the file pointer

- **Opening, Creating, Closing:**

C language provides a number of functions to perform file handling.

fopen() is used to create a new file or open an existing file.

The syntax is as follows:

fp = FILE *fopen(const char *filename, const char *mode);

fp is the file pointer that holds the reference to the file,

the **filename** is the name of the file to be opened or created,

and **mode** specifies the purpose of opening a file such as for reading or writing.

FILE is an object type used for storing information about the file stream.

Example program :

```
#include <stdio.h>
 main ()
  {
     FILE *fp;
     fp = fopen("data.txt", "r");
     if (fp == NULL)
    {
         printf("File does not exist,please check!\n");
     }
     fclose(fp);
  }
```

- **file opening modes:**

Below are some of the most commonly used modes for opening or creating a file.

r	opens a text file in reading mode.
w	opens or creates a text file in writing mode.
a	opens a text file in append mode.
r+	opens a text file in both reading and writing mode. The file must exist.
w+	opens a text file in both reading and writing mode. If the file exists, it's truncated first before overwriting. Any old data will be lost. If the file doesn't exist, a new file will be created.
a+	opens a text file in both reading and appending mode. New data is appended at the end of the file and does not overwrite the existing content.

- **File Close:**

A file needs to be closed after a read or write operation to release the memory allocated by the program.
In C, a file is closed using the fclose() function.
This returns 0 on success and EOF in the case of a failure.
An EOF is defined in the library called stdio.h.
EOF is a constant defined as a negative integer value which denotes that the end of the file has reached.
The syntax is as follows:

fclose(FILE *fp);

Example program :

```
#include <stdio.h>
 main ()
 {
    FILE *fp;
    fp = fopen("data.txt", "r");
    if (fp == NULL)
    {
      printf("File does not exist,please check!\n");
    }
    fclose(fp);
 }
```

- **INPUT/OUTPUT OPERATIONS ON FILES**

➢ **getc(), putc() functions in C:**

getc(), putc() functions are file handling function in C programming language which is used to read a character from a file (getc) and display on standard output or write into a file (putc). Please find below the description and syntax for above file handling functions.

File operation	Declaration & Description
getc()	Declaration: int getc(FILE *fp) getc functions is used to read a character from a file. In a C program, we read a character as below. getc (fp);

putc()	Declaration: int putc(int char, FILE *fp)
	putc function is used to display a character on standard output or is used to write into a file. In a C program, we can use putc as below.
	putc(char,stdout);
	putc(char, fp);

Example:

```
#include <stdio.h>
int main()
{
char ch;
FILE *fp;
if (fp = fopen("test.c", "r"))
{
ch = getc(fp);
while (ch != EOF)
{
putc(ch, stdout);
ch = getc(fp);
}
fclose(fp);
return 0;

}
return 1;
}
```

OUTPUT:
Hi, How are you?

➢ putw(), getw() functions in C

putw(), getw() functions are file handling function in C programming language which is used to write an integer value into a file (putw) and read integer value from a file (getw). Please find below the description and syntax for above file handling functions.

File operation	Declaration & Description
putw()	Declaration: int putw(int number, FILE *fp); putw function is used to write an integer into a file. In a C program, we can write integer value in a file as below. **putw**(i, fp); where i – integer value fp – file pointer
getw()	Declaration: int getw(FILE *fp); getw function reads an integer value from a file pointed by fp. In a C program, we can read integer value from a file as below. getw(fp);

Example:

```
#include <stdio.h>
 int main ()
{
  FILE *fp;
  int i=1, j=2, k=3, num;
  fp = fopen ("test.c","w");
  putw(i,fp);
  putw(j,fp);
  putw(k,fp);
  fclose(fp);
  fp = fopen ("test.c","r");
  while(getw(fp)!=EOF)
  {
    num= getw(fp);
    printf("Data in test.c file is %d \n", num);
  }
```

```
  fclose(fp);
  return 0;
}
```

OUTPUT:
Data in test.c file is
1
2
3

➢ fscanf(), fprintf(), ftell(), rewind() functions in C

fscanf(), fprintf(), ftell(), rewind() functions are file handling functions in C programming language. Please find below the description and syntax for each above file handling functions.

File operation	Declaration & Description
fscanf()	Declaration: int fscanf(FILE *fp, const char *format, …) fscanf() function is used to read formatted data from a file. In a C program, we use fscanf() as below. fscanf (fp, "%d", &age); Where, fp is file pointer to the data type "FILE". Age – Integer variable This is for example only. You can use any specifiers with any data type as we use in normal scanf() function.
fprintf()	Declaration: int fprintf(FILE *fp, const char *format, …) fprintf() function is used to write formatted data into a file. In a C program, we use fprinf() as below. fprintf (fp, "%s %d", "var1", var2); Where, fp is file pointer to the data type "FILE". var1 – string variable var2 – Integer variable This is for example only. You can use any specifiers with any data type as we use in normal printf() function.

ftell()	Declaration: long int ftell(FILE *fp) ftell function is used to get current position of the file pointer. In a C program, we use ftell() as below. **ftell**(fp);
rewind()	Declaration: void rewind(FILE *fp) rewind function is used to move file pointer position to the beginning of the file. In a C program, we use rewind() as below. rewind(fp);

Example:

```
#include <stdio.h>
int main ()
{
  char name [20];
  int age, length;
  FILE *fp;
  fp = fopen ("test.txt","w");
  fprintf (fp, "%s %d", "Fresh2refresh", 5);
  length = ftell(fp); // Cursor position is now at the end of file
  /* You can use fseek(fp, 0, SEEK_END); also to move the
     cursor to the end of the file */
  rewind (fp); // It will move cursor position to the beginning of the file
  fscanf (fp, "%d", &age);
  fscanf (fp, "%s", name);
  fclose (fp);
  printf ("Name: %s \n Age: %d \n",name,age);
  printf ("Total number of characters in file is %d", length);
  return 0;
}
```

OUTPUT:

```
Name: Fresh2refresh
Age: 5
Total number of characters in file is 15
```

➢ fseek(), SEEK_SET, SEEK_CUR, SEEK_END functions in C

fseek() functions is file handling functions in C programming language. It has following constants. SEEK_SET, SEEK_CUR, SEEK_END. Please find below the description and syntax for each above file handling functions.

File operation	Declaration & Description
fseek()	Declaration: int fseek(FILE *fp, long int offset, int whence) fseek() function is used to move file pointer position to the given location. where, fp – file pointer offset – Number of bytes/characters to be offset/moved from whence/the current file pointer position whence – This is the current file pointer position from where offset is added. Below 3 constants are used to specify this.
SEEK_SET	SEEK_SET – It moves file pointer position to the beginning of the file.
SEEK_CUR	SEEK_CUR – It moves file pointer position to given location.
SEEK_END	SEEK_END – It moves file pointer position to the end of file.

Example:

```
include <stdio.h>
int main()
{
   FILE *fp;
   fp = fopen("test.txt", "r");
   // Moving pointer to end
   fseek(fp, 0, SEEK_END);
   // Printing position of pointer
   printf("%ld", ftell(fp));
    return 0;
}
```

OUTPUT:

81

Explanation

The file test.txt contains the following text:
"Someone over there is calling you.
we are going for work.
take care of yourself."

When we implement fseek() we move the pointer by 0 distance with respect to end of file i.e pointer now points to end of the file. Therefore the output is 81.