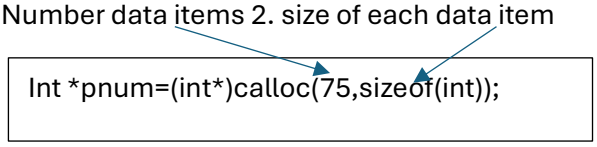
CALLOC

Advantage over malloc

- Allocate mmry as a number of elements of given size
- Initializes the memory that is allocatedso that all bytes are 0;

Two arguments

1.Number data items 2. size of each data item




```
#include <stdio.h>
#include<stdlib.h>
int main()
{
   int *pnum=NULL;
   int num;
   printf("enter the number of elements to be stored:");
   scanf("%d",&num);
   pnum=(int*)calloc(num,sizeof(int));
   for(int i=0;i<num;i++){
      printf("%d>-",pnum[i]);
   }
   return 0;
}
```

```
enter the number of elements to be stored:5
0>-0>-0>-0>-0>-


//here the calloc is initializing every bytes to zero
since no elements are entered
//in  normal compilers malloc will not do this
```

REALLOC

Enables to reuse or extend memory that previously allocated memory using malloc or calloc

Note: preserves the content of the og memory area.

Two arguments

- Pointer containing an address that was previously returned by a call to malloc or calloc
- Size in bytes of new memory that want to be allocated

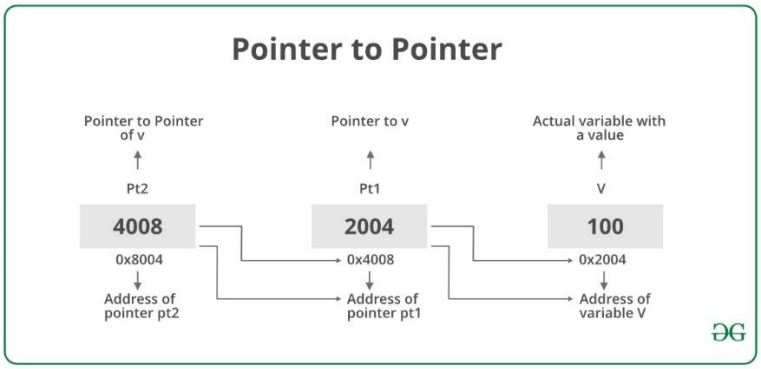
```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
   char *str;
   str=(char*)malloc(15);
   strcpy(str,"antony");
   printf("string=%s,Address=%p\n",str,str);
   str=(char*)realloc(str,25);
   strcat(str,"petta");
   printf("string=%s,Address=%p\n",str,str);
   free(str);
   return 0;
}
```

```
string=antony,Address=0x5bec16cf22a0
string=antonypetta,Address=0x5bec16cf26d0
```

DOUBLE POINTERS -pointer to a pointer

1st pointer contains the address of the value

2nd pointer contains the address of the first pointer




```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
   int **ipp;
   int i=5,j=7,k=6;
   int *ip1,*ip2;
   ip1=&i;
   ip2=&j;
   ipp=&ip1;
   printf("address of i=%p\n",ip1);
   printf("address of j=%p\n",ip2);
   printf("----------------before modification--------------\n");
   printf("ip1=%p\n",ip1);
   printf("ip2=%p\n",ip2);

   // printf("001 i=%d\n",*ip1);
   // printf("002 i=%d\n",**ipp);
   printf("---------------after modification---------------\n");
   *ipp=ip2;//>>initialy *ipp=value in ip1,i.e address of i;>>*ipp=value in ip2;>>ip2 has address of j>>
   printf("ip1=%p\n",ip1);
   printf("ip2=%p\n",ip2);

   return 0;
}
```

```
address of i=0x7ffcce6de4f4
address of j=0x7ffcce6de4f8
----------------before modification--------------
ip1=0x7ffcce6de4f4
ip2=0x7ffcce6de4f8
---------------after modification---------------
ip1=0x7ffcce6de4f8
ip2=0x7ffcce6de4f8
```

Problem 1: Dynamic Array Resizing

Objective: Write a program to dynamically allocate an integer array and allow the user to resize it.

Description:

1. The program should ask the user to enter the initial size of the array.
2. Allocate memory using malloc.
3. Allow the user to enter elements into the array.
4. Provide an option to increase or decrease the size of the array. Use realloc to adjust the size.
5. Print the elements of the array after each resizing operation.

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

  int size;
  printf("enter the size of array :");
  scanf("%d",&size);
  int *p_arr=NULL;
  p_arr=(int*)malloc(size*sizeof(int));
  printf("enter the array elements\n");
  for(int i=0;i<size;i++){
    scanf("%d",&p_arr[i]);
  }
  for(int i=0;i<size;i++){
    printf("%d>-",p_arr[i]);
  }
  //increase size
  int i_size;
  printf("\nenter size to be increased to:");
  scanf("%d",&i_size);
  printf("after increasing\n");
  p_arr=(int*)realloc(p_arr,i_size*sizeof(int));
  for(int i=size;i<i_size;i++){
    scanf("%d",&p_arr[i]);
  }
  for(int i=0;i<i_size;i++){
    printf("%d>-",p_arr[i]);
  }
  //decrease size
  printf("\nenter size to be decreased to\n ");
  int d_size;
  scanf("%d",&d_size);
  printf("after decreasing\n");
  p_arr=(int*)realloc(p_arr,d_size*sizeof(int));
  for(int i=size;i<d_size;i++){
    scanf("%d",&p_arr[i]);
  }
  for(int i=0;i<d_size;i++){
    printf("%d>-",p_arr[i]);
  }
  return 0;
}
```

Problem 2: String Concatenation Using Dynamic Memory

Objective: Create a program that concatenates two strings using dynamic memory allocation.

Description:

1. Accept two strings from the user.
2. Use malloc to allocate memory for the first string.
3. Use realloc to resize the memory to accommodate the concatenated string.
4. Concatenate the strings and print the result.
5. Free the allocated memory.

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

   char *p_str1;
   p_str1=(char*)malloc(20*sizeof(char));
   printf("enter string 1\n");
   scanf("%[^\n]",p_str1);
   char *p_str2;
   p_str2=(char*)malloc(20*sizeof(char));
   printf("enter string 2\n");
   scanf(" %[^\n]",p_str2);
   p_str1=(char*)realloc(p_str1,(sizeof(*p_str2))+1);
   strcat(p_str1," ");
   strcat(p_str1,p_str2);
   printf("%s",p_str1);
   free(p_str1);
   free(p_str2);
   return 0;
}
```

Problem 3: Sparse Matrix Representation

Objective: Represent a sparse matrix using dynamic memory allocation.

Description:

1. Accept a matrix of size m×nm \times nm×n from the user.
2. Store only the non-zero elements in a dynamically allocated array of structures (with fields for row, column, and value).
3. Print the sparse matrix representation.
4. Free the allocated memory at the end.

Problem 4: Dynamic Linked List Implementation

Objective: Implement a linked list using dynamic memory allocation.

Description:

1. Define a struct for linked list nodes. Each node should store an integer and a pointer to the next node.
2. Create a menu-driven program to perform the following operations:
    - Add a node to the list.
    - Delete a node from the list.
    - Display the list.
3. Use malloc to allocate memory for each new node and free to deallocate memory for deleted nodes.

Problem 5: Dynamic 2D Array Allocation

Objective: Write a program to dynamically allocate a 2D array.

Description:

1. Accept the number of rows and columns from the user.
2. Use malloc (or calloc) to allocate memory for the rows and columns dynamically.
3. Allow the user to input values into the 2D array.
4. Print the array in matrix format.
5. Free all allocated memory at the end.

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

  int row,col;
  int *matrix=NULL,*p_col=NULL;
  printf("enter the row and col size\n");
  scanf("%d %d",&row,&col);
  matrix=(int*)malloc(row*col*sizeof(int));
  printf("enter the elements\n");

  for(int i=0;i<row*col;i++){
    scanf("%d",&matrix[i]);

  }
  for(int i=0;i<row;i++){
    for(int j=0;j<col;j++){
      printf("%d",matrix[j]);
    }
    printf("\n");
  }
  free(matrix);

  return 0;
}
```

****

STRUCTURES

Syntax

```
Struct date
{
int month;
int day;
int year;
};
```

NOTE: no memory is allocated for the declaration;

Act as a blueprint

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

```
#include<stdlib.h>
#include<string.h>
struct date{
   int day;
   int month;
   int year;
};
/*
struct date{
   int day;
   int month;
   int year;
}today;
*/
int main()
{
   struct date today;

   today.day=21;
   today.month=11;
   today.year=2024;
   struct date tmrw={21,11,2024};
   printf("todays date is %d-%d%d\n", today.day,today.month,today.year);
   printf("size of today=%ld\n",sizeof(today));
   printf("tomorows date is %d-%d-%d", today.day,today.month,today.year);
   return 0;
}
```

```
#include <stdio.h>
struct Coordinate{
   int x;
   int y;
};
void printCoordinate(struct Coordinate);
int main(){
   printCoordinate((struct Coordinate){5, 6});
   /*struct Coordinate pointA = {5,6};
   printCoordinate(pointA);*/
   return 0;
}
void printCoordinate(struct Coordinate temp){
   printf("x = %d  y = %d \n",temp.x, temp.y);
}
```

Array of structurs

```
#include<stdio.h>
struct coordinate{
   int x;
   int y;
};
int main(){
   struct coordinate arr_c[5];
   for(int i=0;i<5;i++){
```

```
    printf("enter x and y coordinate :\n");
    scanf("%d %d",&arr_c[i].x,&arr_c[i].y);
  }
   for(int i=0;i<5;i++){
    printf("x=%d y=%d\n",arr_c[i].x,arr_c[i].y);

  }
}
```

Array in structure

```
#include<stdio.h>
struct month{
  int no_days;
  char name[3];
};
int main(){
  struct month allMonths[12];
  for(int i=0;i<12;i++){
    printf("%d month and number of days\n",i+1);
    scanf("%s %d",allMonths[i].name,&allMonths[i].no_days);
  }
  for(int i=0;i<12;i++){
    printf("%s has %d days\n",allMonths[i].name,allMonths[i].no_days);

  }
}
```

Nested structure

```
#include<stdio.h>
struct Date{
  int day;
  int month;
  int year;
};
struct Time{
  int sec;
  int min;
  int hour;
};
struct DateTime{
  struct Date d1;
  struct Time t1;
};
int main(){
  struct DateTime dt={{21,11,2024},{51,01,17}};


  printf("current date=%d-%d-%d\n",dt.d1.day,dt.d1.month,dt.d1.year);
  printf("current time=%d-%d-%d",dt.t1.hour,dt.t1.min,dt.t1.sec);
  return 0;
}
```

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

```
#include <string.h>
struct student {
  char name[50];
  int rollNum;
  float marks;
};
void addStudent(struct student[],int);
void printStudentData(struct student[],int);
int findByRoll(struct student[],int,int);
float avgMarks(struct student[],int);
int main(){
  int no,option;


  struct student stdDetails[no];
  int flag = 0;
  printf("Enter your choice: ");
  scanf("%d", &option);
  switch(option){
    case 1:
      printf("enter number of student details to be entered:");
      scanf("%d",&no);
      addStudent(stdDetails,no);
      flag=1;
      break;
    case 2:
      if(flag==0){
        printf("no data enterd.\n");
        return 0;
      }
      printStudentData(stdDetails,no);
      break;
    case 3:
      if(flag==0){
          printf("no data enterd.\n");
          return 0;
        }
      int rollToSrch;
      printf("enter roll numbe to srch:");
      scanf("%d",&rollToSrch);
      int roll=findByRoll(stdDetails,no,rollToSrch);
      if(roll>=0&&roll<no){
        printf("student found\n");
        printf("%s || %d || %f\n",stdDetails[roll].name,stdDetails[roll].rollNum,stdDetails[roll].marks);
      }
      else{
        printf("roll number not found\n");
        printf("student not found\n");
      }
      break;
    case 4:
      if(flag==0){
        printf("no data enterd.\n");
        return 0;
```

```
        }
        float avg=avgMarks(stdDetails,no);
        printf("average marks of %d students is %f",no,avg);
        break;
      default:
        printf("invalid option");
    }
}
void addStudent(struct student stdDetails[],int no) {
    for(int i=0;i<no;i++){
      printf("enter details of %d student\n",i+1);
      scanf(" %[^\n] %d %f",stdDetails[i].name,&stdDetails[i].rollNum,&stdDetails[i].marks);
    }
}
void printStudentData(struct student stdDetails[],int no){
    for(int i=0;i<no;i++){
      printf("%s || %d || %f\n",stdDetails[i].name,stdDetails[i].rollNum,stdDetails[i].marks);
    }
}
int findByRoll(struct student stdDetails[],int no,int Roll){
    int i=0;
    while(i<no){
      if(stdDetails[i].rollNum==Roll){
        return i;
      }
      else{
        i++;
      }
    }
    return i;
}
float avgMarks(struct student stdDetails[],int no){
    float sum=0,avg;
    for(int i=0;i<no;i++){
      sum=sum+stdDetails[i].marks;
    }
    avg=sum/no;
    return avg;
}
```

Problem 1: Employee Management System

Objective: Create a program to manage employee details using structures.

Description:

1. Define a structure Employee with fields:
    - int emp_id: Employee ID
    - char name[50]: Employee name
    - float salary: Employee salary
2. Write a menu-driven program to:
    - Add an employee.
    - Update employee salary by ID.
    - Display all employee details.
    - Find and display details of the employee with the highest salary.

Problem 2: Library Management System

Objective: Manage a library system with a structure to store book details.

Description:

1. Define a structure Book with fields:
    - int book_id: Book ID
    - char title[100]: Book title
    - char author[50]: Author name
    - int copies: Number of available copies
2. Write a program to:
    - Add books to the library.
    - Issue a book by reducing the number of copies.
    - Return a book by increasing the number of copies.
    - Search for a book by title or author name.

Problem 3: Cricket Player Statistics

Objective: Store and analyze cricket player performance data.

Description:

1. Define a structure Player with fields:
    - char name[50]: Player name
    - int matches: Number of matches played
    - int runs: Total runs scored
    - float average: Batting average
2. Write a program to:
    - Input details for n players.
    - Calculate and display the batting average for each player.
    - Find and display the player with the highest batting average.

Problem 4: Student Grading System

Objective: Manage student data and calculate grades based on marks.

Description:

1. Define a structure Student with fields:
    - int roll_no: Roll number
    - char name[50]: Student name
    - float marks[5]: Marks in 5 subjects
    - char grade: Grade based on the average marks
2. Write a program to:
    - Input details of n students.
    - Calculate the average marks and assign grades (A, B, C, etc.).
    - Display details of students along with their grades.

Problem 5: Flight Reservation System

Objective: Simulate a simple flight reservation system using structures.

Description:

1. Define a structure Flight with fields:
    - char flight_number[10]: Flight number
    - char destination[50]: Destination city

- int available_seats: Number of available seats

2. Write a program to:
    - Add flights to the system.
    - Book tickets for a flight, reducing available seats accordingly.
    - Display the flight details based on destination.
    - Cancel tickets, increasing the number of available seats.