**DATA STRUCTURE**

**Data:** Any piece of information is called data.

**Structure:** Structure refers to the way of representing of data.

**Data Structure**

Data Structure is a concept of representing the data in memory on different format.

Data Structure is a concept of representation of relationship between logical related data elements. It is an arrangement of data in computer memory.

In data structure, decision on the operation such as storage, retrieval and access must be carried between logically related data elements only.

**Applications**

1. Compiler Designing
2. Operating System
3. Data Base Management System
4. Numerical Analysis
5. Graphics
6. Artificial Intelligence
7. Simulation
8. Statistical Analysis Package

**Classification**

Data Structure is classified into two types:

1. Linear Data Structure
2. Non – Linear Data Structure

**Linear Data Structure**

A data structure is called linear, when the elements are stored in sequential manner.

In linear data structure, a sequential relationship will be exist between elements.

**Example**

Array, stack, queue, linked list

**Non Linear data structure**

In this data structure, elements are arranged in non – linear manner.

**Example**

Trees, graphs, tables

**Memory Allocation in C**

In C programming language, there are two types of memory allocations are possible:

1. Static memory allocation(compile – time memory allocation)
2. Dynamic memory allocation (run – time memory allocation)

**Static memory allocation**

In implementation, whenever we are creating memory at compile time, then it is called static memory allocation.

When we are working with static memory allocation, if the allocated memory is not sufficient then we cannot extend it.

Using compile time memory management, we cannot handle the memory or we cannot utilize memory efficiently.

In implementation, when we need to use memory more efficiently then go for dynamic memory allocation.

**Dynamic Memory Allocation**

It is a concept of allocating or de – allocating memory at run time.

Using dynamic memory allocation, we can utilize memory more efficiently depending on requirement i.e. how much memory we are required that much memory we can allocate and de – allocation is also dependent on us.

By using dynamic memory allocation, whenever we want, which type we want and how much we want that time, that type and that much memory we can construct dynamically.

Dynamic memeory allocation related functions are declared in **<alloc.h>** or **<malloc.h>** or **<mem.h>**.

DMA related functions are:

1. malloc() : 1 argument required
2. calloc() : 2 arguments required
3. realloc() : 2 arguments required
4. free() : 1 argument required

**malloc()**

By using malloc(), we can create memory dynamically at initial stage.

malloc() function is required one argument of type size\_type i.e. data type size.

malloc() function will create memory in byte format and initial value of the memory location is garbage.

**Syntax**

void \* malloc(size\_type);

**calloc()**

By using calloc() function, we can create the memory dynamically at initial stage.

calloc() function required two arguments of type count, size\_type.

calloc() function will create memory in block format and initial value is zero (0).

**Syntax**

void \* calloc(count, size\_type);

**realloc()**

By using realloc() function, we can create memory dynamically at middle or ending stage.

realloc() function required two arguments i.e. void\* and size\_type.

**Syntax**

void \* realloc(void \*, size\_type);

Here, void \* gives previous block address and size\_type is data – type size.

**free()**

By using free memory, we can de – allocate the memory which is allocated dynamically.

free() required on argument which is of type **void \***.

**Syntax**

void free(void \*);

Dynamic memory Allocation depends on three factors:

1. time
2. type
3. size

**Example**

int \* a;

a = (int \*) malloc (sizeof(int)); // 2 bytes

double \* a;

a = (double \*) malloc(sizeof(double)); // 8 bytes

int \* a;

a = (int \*) calloc(5,sizeof(int)); // 10 bytes

char \* a;

a = (char \*) calloc(50,sizeof(char)); // 50 bytes

When we are working with dynamic memory allocation related function, then we can apply for any data – type, so type casting process should be required because it will create type less or byte data only.

If the memory is constructed element by element, then it is called byte format representation and if entire memory is constructed in a single call then it is called block format representation.

**Program to create array size dynamically, enter elements and find the sum.**

#include <stdio.h>

#include<conio.h>

#include<alloc.h>

int main()

{

int \*a;

int n , i , sum = 0;

clrscr();

printf("Enter size of the array:");

scanf("%d",&n);

a = ( int \* ) malloc ( sizeof(int) \* n);

if ( a == NULL )

{

printf("Memory not allocated");

getch();

return 0;

}

else

{

for ( i = 0; i < n; i++)

{

printf("Enter the element:");

scanf("%d",&a[i]);

sum += a[i];

}

}

printf("Sum = %d" , sum);

getch();

free(a);

a = NULL;

return 0;

}

Output

Enter size of the array: 5

Enter the element: 1

Enter the element: 2

Enter the element: 3

Enter the element: 4

Enter the element: 5

Sum = 15

Enter size of the array: 0

Memory not allocated

**Note:** In the above program, in place of **int main()** if we will use **void main()**, then compulsory we have to give **exit(1)** function inside if statement, because void main() does not provide exit status of application whereas int main() provide exit status of application.