Datas structure operations

There are six basic operations that can be perfor-med on data structure.

(a) Traversing
(b) searching
(c) Sorting
(d) Inserting
(e) Deleting
(f) Merging

(a) Traversing

- Traversing means accessing and processing each ele-ement in the data structure exactly once.

- This operation is used for counting the number of element, printing the contents of the element etc.

(b) Searching

- searching in finding out the location of a given element from a set of numbers.

(c) Sorting

- sating is the process of arranging a list of element in a sequential order.

- The sequential order may be a descending order or an ascending order according to the requirements of data structure.

(d) Inserting
- Inserting an element is adding an element in the data structure at any location. After insert operation the number of elements are increased by one.

(e) Deleting
- Deleting an element is removing an element in the data structure at any position. After deletion the number of elements are deceased by one.

(f) Merging
- The process of combining the elements of two data structures into a single data structure is called Merging.

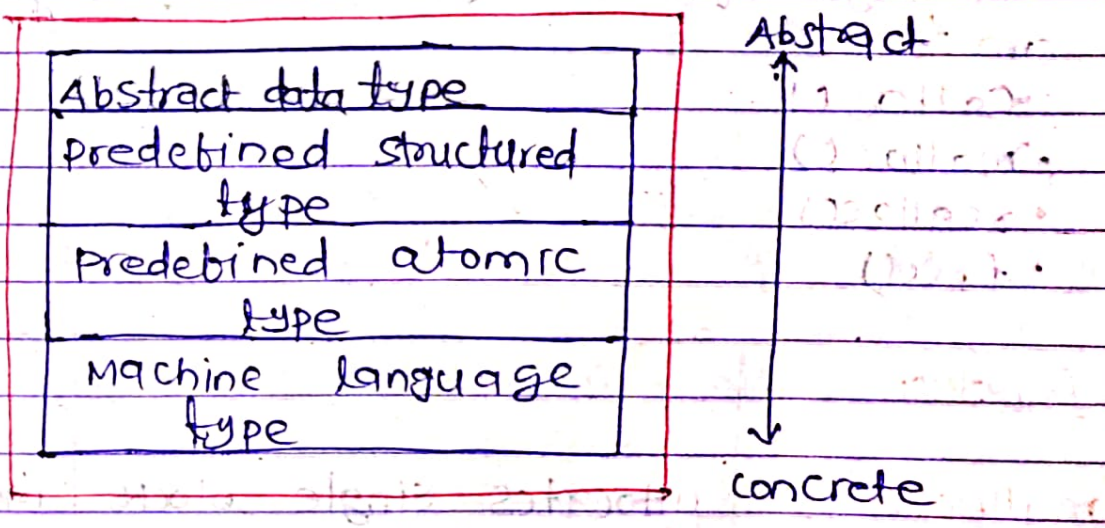
Abstract data type (ADT)
An ADT consist of a data type together with a set of operations, which define how the the type may be manipulated.

The abstract datatype is a special datatype, whose behaviour is defined by a set of values and set of operations.

Abstract datatype exist conceptually and concentrate on the mathematical properties of the data type ignoring implementation constraints and details. ADT is a type for objects whose behaviour is defined by a set of value and a set of operation.

The definition of ADT only mentions what operation are to be performed but not how these operation will be implemented. It doesn't specify how data will be organized in memory and what algorithm will be used for implementing the operations.

It is called Abstract, because it gives an implementation in dependent view. The process of providing only the essentials and hiding the details is known as abstraction.



Dynamic Memory allocation in C

The concept of dynamic memory allocation in C language enables the C programmer to allocate memory at runtime.

Static Memory allocation	Dynamic Memory allocation
⇒ Memory is allocated at compile time.	⇒ Memory is allocated at run time.
⇒ Memory cannot be increased while executing program.	⇒ Memory can be increased while executing program.
⇒ It is done by using an array.	⇒ It is done by using linked list

⇒ C language offers dynamic memory allocation. They are :-

- Calloc()
- malloc()
- realloc()
- free()

function	uses	Syntax
malloc()	allocates single block of requested memory.	malloc(number * sizeof(int));
Calloc()	allocates multiple block of requested memory	calloc(number, sizeof(int));
realloc()	reallocate the memory occupied by malloc() and calloc() functions.	realloc(pointer_name, number * sizeof(int));

free()	Frees the dynamically allocated memory	free(pointer_name);

1. malloc() function

- "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.

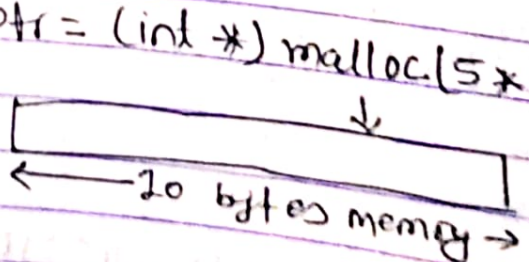
Syntax

ptr = (int *) malloc(5 * sizeof(int));

since, the size of int is 2 bytes, this statement will allocate 10 bytes of memory. And the pointer ptr holds the address of the first byte in the allocated memory.

malloc()

int *ptr = (int *) malloc(5 * sizeof(int));



Example program to illustrate malloc() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int *ptr;
 int n, i, sum=0;
 clrscr();
 printf("enter number of elements\n");
 scanf("%d",&n);
 ptr=(int *)malloc(n*sizeof(int));

 if(ptr==NULL)
 {
   printf("Memory not allocated\n");
   exit(0);
 }

 else
 {
   for(i=0;i<n;i++)
   {
     ptr[i]=i+1;
   }
 }
 printf("The elements are:\n");
 for(i=0;i<n;i++)
 {
```

```
    printf("%d\n", ptr[i]);
}
getch();
}
```

O/P: Enter number of element
5
The elements are
1
2
3
4
5

Taking input from user, we will make certain changes in above program from else block.

```
else
{
  printf("Enter %d numbers\n", n);
  for(i=0; i<n; i++)
  {
    scanf("%d", &num);
    ptr[i]= num;
  }
printf("The elements are: \n");
```

```
for(i=0 ; i<n ; i++)
{
    printf("%d \n", ptr[i]);
}
getch();
}
```

2. calloc() method

→ "calloc" or "contiguous allocation" method in C is used to dynamically allocate the specified number of blocks memory of the specified type.

→ It initializes each block with a default value '0'

Syntax:

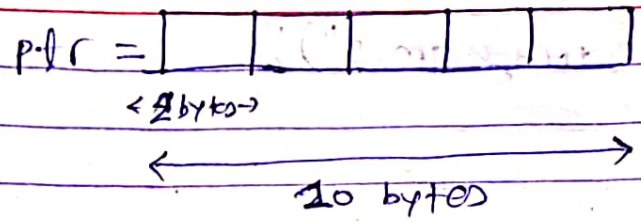
```
ptr = (cast_type *) calloc(n, element_size);
```

For eg: `ptr(float *) calloc(25, sizeof(float));`

This statement allocates contiguous space in memory 25 elements each with the size of the float.

calloc()

```
int *ptr = (int *) calloc(5, sizeof(int));
```



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

Example program to illustrate calloc() function

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int *ptr;
  int n, i;
  printf("Enter number of elements : %d\n", n);
  scanf("%d", &n);
  ptr= (int*)calloc(n, sizeof(int));
  if(ptr==NULL)
  {
   Printf("Memory not allcated\n");
   exit(0);
  }
  else
  {
   for(i=0; i<n; i++)
   {
     ptr[i] = i+1;
   }
}
```

Printf("The elements of the array are:");
for

3. Free()

→ free() method in C is used to dynamically deallocate the memory.

→ The memory allocated using function malloc() and calloc() is not deallocated 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()~~ 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 insufficient, realloc() can be used to dynamically re-allocate memory.

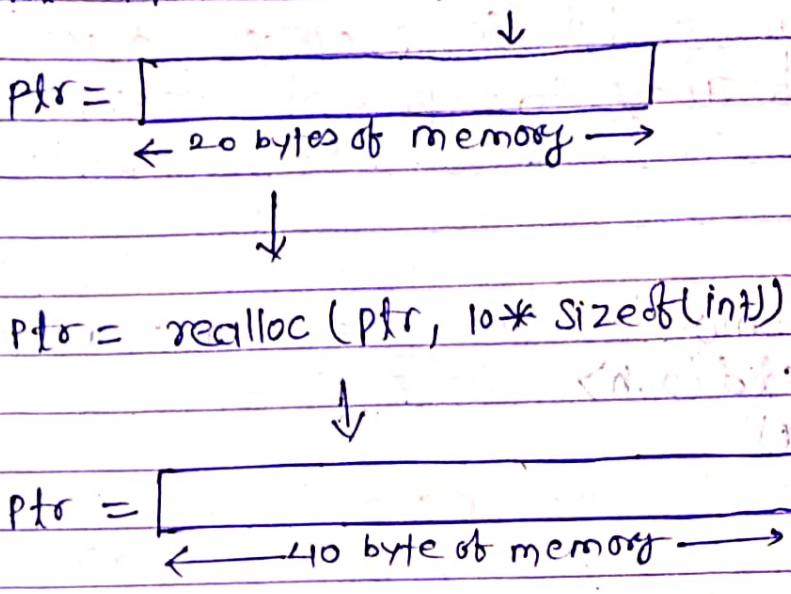
Syntax:

ptr = realloc(ptr, newsize)

where, ptr is reallocated with new size "newsize".

realloc()

```
int *ptr = (int *) malloc(5 * sizeof(int));
```




```
ptr = realloc (ptr, 10 * sizeof(int))
```


Example program to illustrate realloc() function.

```
#include<conio.h>
#include<stdio.h>
void main()
{
char *name;
name = (char*) malloc(10);
strcpy(name," D.K. yadav");
printf("\n Name= %s", name);
name= (char *) realloc(name,30);
strcpy(name," Dilip Kumar Yadav");
printf("\n Name = %s", name);
getch();
}
```

O/P: D.K Yadav
Dilip Kumar Yadav

Write a program to read number of student from user and then read mark of each student. Display entered mark and their average value. Use pointer instead of array to represent marks of different student.

```
#include<conio.h>
#include<stdio.h>
void main()
{
  int n, i;
  float *p, sum=0, avg=0;
  clrscr();
  printf("How many students are there? \t");
  scanf("%d", &n);
  printf("Enter marks of each students \n");
  p=(float*)malloc(n*sizeof(float));
  for(i=0; i<n; i++)
  {
  }
  {
    scanf("%f", p+i);
    sum+= *(p+i);
  }
  avg=sum/n;

  printf("\n The average mark of it");
```

```
for(i=0 ; i<n, i++)
{
  printf("\t %.2f ", *(p+i));
}
Printf(" IS : %.2f", avg);

free(p);
getch();
}
```

O/P: How many students are there ? 5
Enter mark of each student
45.5, 56 89 90.5 47

The average marks of 45.50 56.00 89.00 90.50 47.00
is : 65.60

Difference between calloc() and malloc()

There are two major differences between malloc and calloc in c programming language: First, in the number of arguements. The malloc() takes a single argueiment, while calloc() takes two. second, malloc() doesn't initialize the memory allocated, while calloc() initializes the allocated memory to zero.

Another

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

```
void main()
{
int s, *p, *ptr, i, sum=0, *q;
clrscr(); enter size for Array
printf("enter number of elements\n");
scanf("%d", &s);
ptr = (int *) malloc(s*sizeof(int));
p=ptr;
printf("Memory allocated [redacted] %d", ptr);
if(ptr==NULL)
{
   printf("out of Memory \n");
exit(1);
}

else printf("enter %d elements", s);
{   for (i=1; i<s; i++)
    {
       scanf("%d", ptr);
       sum += *ptr;
       ptr++;
    }

Printf("Elements are :");
for( i=0; i<s; i++)
{
   printf("%d", [redacted]*p);
   p++;
}
printf("The addition is %d", sum);
```

```
printf("Enter new size for Array \n");
scanf("%d", &s)
ptr = (int *) realloc (ptr, s * sizeof(int));
printf("memory
if (ptr == NULL)
{
  printf("out of memory");
  exit(0);
}
printf("Reallocated memory %d ", ptr);
q = ptr;
printf ("enter %d elements \n", s);
for (i=0 ; i<=s ; i++)
{
   scanf("%d", ptr);
   sum += *ptr;
   ptr++;
}
printf("Elements are :")
for (i=0; i<s ; i++)
{
   printf("%d", *q);
   q++;
}
printf("The addition is %d", sum);
getch();
}
```