18-06-2024

DYNAMIC MEMORY ALLOCATION FUNCTIONS :-

-> malloc( ) -> calloc( ) -> realloc( )

1. malloc( ) :-

SYNTAX:-

(type *) malloc (size in bytes)

-> It allocates memory of size in bytes

-> Allocated memory is always in continuous memory locations & contains initial Garbage Values.

-> If it is successful to allocate memory returns VOID pointer so typecasting is needed.

-> If it is not successful to allocate memory returns NULL pointer [0; 10; NULL]

Ex:- Reading & Printing of n-elements using Dynamic memory allocation

```
-> # include <stdio.h>
main ()
{
  int i, n, *p;
  printf ("enter n");
  scanf ("%d", &n);
  p = (int *) malloc (n * sizeof (int));
  if (p = = NULL)
{
  printf ("memory not available");
  exit (0);
}
printf ("enter elements");
for (i=0; i<n; i++)
scanf ("%d", p+i);
  for (i=0; i<n; i++)
printf("%d", *(p+i));
  free (p);
}
```

O/P :-

```
enter n: 3
enter elements: 1 2 3
1 2 3
```

2. calloc () :-

SYNTAX:-

```
(type *) calloc (n, size in bytes)
```

-> It allocates 'n' rows of size in bytes.

-> Allocated memory contains initial zero's

-> If it is successful to allocate memory it returns VOID pointer, so typecasting is needed.

-> If it is not successful to allocate memory it returns Null pointer.

Ex:-

```
struct Details
{
  char name [20];
  int age;
  float height;
};
(struct Details *) calloc (n, sizeof (struct Details));
```

3. realloc ( ) :-

SYNTAX :-

(type *) realloc (old pointer, new size)

-> It reallocates memory of new size & transfers old data into newly allocated space.

Ex :-
```
# include < stdio.h >
main ( )
{  char *p = " sree ", *q;
   q = ( char * ) realloc ( p , 20 );
   strcat ( q , " Hari " );
   puts ( q );
   free ( q );
}
```

ADVANTAGES OF POINTERS =>
-> Fast execution
-> A lot of memory is saved during string handling
-> Dynamic memory allocation
-> By call by address we can call any number of values.

DRAWBACKS OF POINTERS =>
-> Lot of confusion
-> low security

COMMAND LINE ARGUMENTS :-

Ex:-

```
-> #include <stdio.h>
   void main (int argc, char* argv[])
   {
       int a, b, c;
       printf ("enter a,b");
       scanf ("%d %d", &a, &b);
       c = a+b;
       printf ("%d", c);
       printf ("%s %s", argv[1], argv[2]);
   }
```

getch the char

UNFORMATED CHARACTER READING FUNCTIONS ( ):-

1. getch ( ):- It doesnot echo the character & doesnot need "ENTER" key to read the character.
2. getche ( ):- It echos the character & doesnot ~~need~~ read "ENTER" key to read the character.
3. ~~char~~ -> getchar ( ):- It echos the character & reads "ENTER" key to read the character.

Ex:-
```
main ( )
{
    char x;
    printf ("enter char");
    x = getchar ( );
    putchar (x);
}
```