**Management of Memory:**

Almost all computer languages can handle system memory. All the variables used in your program occupies a precise memory space along with the program itself, which needs some memory for storing itself (i.e., its own program). Therefore, managing memory utmost care is one of the major tasks a programmer must keep in mind while writing codes.

When a variable gets assigned in a memory in one program, that memory location cannot be used by another variable or another program. So, C language gives us a technique of allocating memory to different variables and programs.

There are two types used for allocating memory. These are:

**Static memory allocations:**

In the static memory allocation technique, allocation of memory is done at compilation time, and it stays the same throughout the entire run of your program. Neither any changes will be there in the amount of memory nor any change in the location of memory.

**Dynamic memory allocations:**

In dynamic memory allocation technique, allocation of memory is done at the time of running the program, and it also has the facility to increase/decrease the memory quantity allocated and can also release or free the memory as and when not required or used. Reallocation of memory can also be done when required. So, it is more advantageous, and memory can be managed efficiently.

***malloc***, ***calloc***, or ***realloc*** are the three functions used to manipulate memory. These commonly used functions are available through the ***stdlib*** library so you must include this library to use them.

## C - Dynamic memory allocation functions

|  |  |
| --- | --- |
| Function | Syntax |
| malloc() | malloc (number \*sizeof(int)); |
| calloc() | calloc (number, sizeof(int)); |
| realloc() | realloc (pointer\_name, number \* sizeof(int)); |
| free() | free (pointer\_name); |

## malloc function

* malloc function is used to allocate space in memory during the execution of the program.
* malloc function does not initialize the memory allocated during execution.  It carries garbage value.
* malloc function returns null pointer if it couldn't able to allocate requested amount of memory.

### Example program for malloc() in C

Example:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

char \*mem\_alloc;

/\* memory allocated dynamically \*/mem\_alloc = malloc( 15 \* sizeof(char) );

if(mem\_alloc== NULL )

{

printf("Couldn't able to allocate requested memory\n");

}

else

{

strcpy( mem\_alloc,"w3schools.in");

}

printf("Dynamically allocated memory content : %s\n", mem\_alloc );

free(mem\_alloc);

}

## calloc function

* calloc () function and malloc () function is similar. But calloc () allocates memory for zero-initializes. However, malloc () does not.

### Example program for calloc() in C

Example:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

char \*mem\_alloc;

/\* memory allocated dynamically \*/mem\_alloc = calloc( 15, sizeof(char) );

if( mem\_alloc== NULL )

{

printf("Couldn't able to allocate requested memory\n");

}

else

{

strcpy( mem\_alloc,"w3schools.in");

}

printf("Dynamically allocated memory content : %s\n", mem\_alloc );

free(mem\_alloc);

}

**realloc function**

* realloc function modifies the allocated memory size by malloc and calloc functions to new size.
* If enough space doesn't exist in the memory of current block to extend, a new block is allocated for the full size of reallocation, then copies the existing data to the new block and then frees the old block.

## free function

* free function frees the allocated memory by malloc (), calloc (), realloc () functions.

### Example program for realloc() and free()

Example:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

char \*mem\_alloc;

/\* memory allocated dynamically \*/mem\_alloc = malloc( 20 \* sizeof(char) );

if( mem\_alloc == NULL )

{

printf("Couldn't able to allocate requested memory\n");

}

else

{

strcpy( mem\_alloc,"w3schools.in");

}

printf("Dynamically allocated memory content : " \ "%s\n", mem\_alloc );

mem\_alloc=realloc(mem\_alloc,100\*sizeof(char));

if( mem\_alloc == NULL )

{

printf("Couldn't able to allocate requested memory\n");

}

else

{

strcpy( mem\_alloc,"space is extended upto 100 characters");

}

printf("Resized memory : %s\n", mem\_alloc );

free(mem\_alloc);

}

Resized memory: space is extended up to 100 characters

**Unions:**

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 a Union in C

Syntax:

union unionName

{

//member definitions

};

Example:

union Courses

{

char WebSite[50];

char Subject[50];

int Price;

};

Accessing Union Members in C

Example:

#include<stdio.h>

#include<string.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);

}

Program Output:

Book Author: The C Programming Language

Book Price: 0

**Typedef:**

C is such a dominant language of its time and now, that even you can name those primary data type of your own and can create your own named data type by blending data type and its qualifier.

## The typedef keyword in C

**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>;

Here's a sample code snippet as of how typedef command works in C:

Example:

typedef signed long slong;

slong in the statement as mentioned above is used for a defining a signed qualified long kind of data type. Now the thing is this 'slong', which is an user-defined identifier can be implemented in your program for defining any signed long variable type within your C program. This means:

Example:

slong g, d;

will allow you to create two variables name 'g' and 'd' which will be of type signed long and this quality of signed long is getting detected from the slong (typedef), which already defined the meaning of slong in your program.

### Various Application of typedef

The concept of typedef can be implemented for defining a user-defined data type with a specific name and type. This typedef can also be used with structures of C language. Here how it looks like:

Syntax:

typedef struct

{

type first\_member;

type sec\_member;

type thrid\_member;

} nameOfType;

Here nameOfType correspond to the definition of structure allied with it. Now, this nameOfType can be implemented by declaring a variable of this structure type.

nameOfType type1, type2;

Simple Program of structure in C with the use of typedef:

Example:

#include<stdio.h>

#include<string.h>

typedef struct professor

{

char p\_name[50];

int p\_sal;

} prof;

void main(void)

{

prof pf;

printf("\n Enter Professor details: \n \n");

printf("\n Enter Professor name:\t");

scanf("% s", pf.p\_name);

printf("\n Enter professor salary: \t");

scanf("% d", &pf.p\_sal);

printf("\n Input done ! ");

}

### Using typedef with Pointers

typedef can be implemented for providing a pseudo name to pointer variables as well. In this below-mentioned code snippet, you have to use the typedef, as it is advantageous for declaring pointers.

int\* a;

The binding of pointer (\*) is done to the right here. With this kind of statement declaration, you are in fact declaring an as a pointer of type int (integer).

typedef int\* pntr;

pntr g, h, i;