C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;

int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type in teger.
```

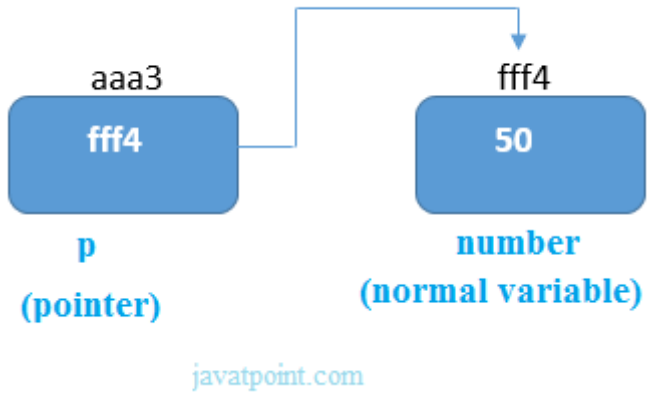
Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
int *a;//pointer to int
char *c;//pointer to char
```

Pointer Example

An example of using pointers to print the address and value is given below.



As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), we can print the value of pointer variable p.

Important point to note is: The data type of pointer and the variable must match, an int pointer can hold the address of int variable, similarly a pointer declared with float data type can hold the address of a float variable. In the example below, the pointer and the variable both are of int type.

```
#include <stdio.h>
int main()
{
  //Variable declaration
  int num = 10;

  //Pointer declaration
  int *p;

  //Assigning address of num to the pointer p
  p = &num;

  printf("Address of variable num is: %u", p);
  return 0;
}
```

Output

```
Address of variable num is: 0x7fff5694dc58
```

C Pointers – Operators that are used with Pointers

"Address of"(&) Operator

I have used &num to access the address of variable num. The **& operator** is also known as "**Address of**" Operator.

```
printf("Address of var is: %u", &num);
```

Pointer is just like another variable; the main difference is that it stores address of another variable rather than a value.

"Value at Address"(*) Operator

The * Operator is also known as **Value at address** operator.

How to declare a pointer?

```
int *p1  /*Pointer to an integer variable*/
double *p2  /*Pointer to a variable of data type double*/
char *p3   /*Pointer to a character variable*/
float *p4   /*pointer to a float variable*/
```

The above are the few examples of pointer declarations. **If you need a pointer to store the address of integer variable then the data type of the pointer should be int**. Same case is with the other data types.

By using * operator we can access the value of a variable through a pointer.
For example:

```
double a = 10;
double *p;
p = &a;
```

*p would give us the value of the variable a. The following statement would display 10 as output.

```
printf("%d", *p);
```

Similarly if we assign a value to *pointer like this:

```
*p = 200;
```

It would change the value of variable a. The statement above will change the value of a from 10 to 200.

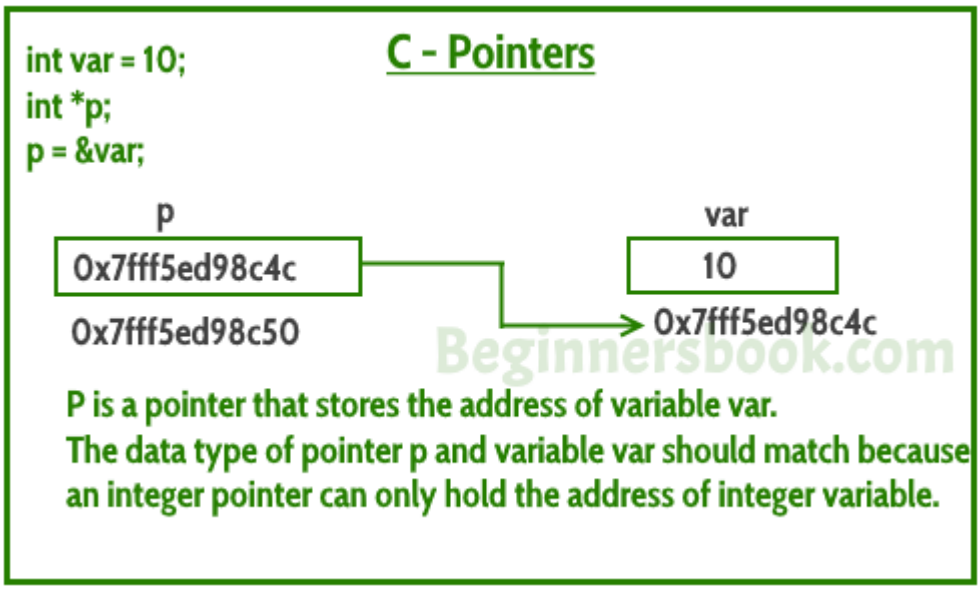
Example of Pointer demonstrating the use of & and *

```
#include <stdio.h>
int main()
{
  /* Pointer of integer type, this can hold the
   * address of a integer type variable.
   */
  int *p;

  int var = 10;

  /* Assigning the address of variable var to the pointer
   * p. The p can hold the address of var because var is
   * an integer type variable.
   */
  p= &var;
```

```
  printf("Value of variable var is: %d", var);
  printf("\nValue of variable var is: %d", *p);
  printf("\nAddress of variable var is: %p", &var);
  printf("\nAddress of variable var is: %p", p);
  printf("\nAddress of pointer p is: %p", &p);
  return 0;
}
```

Output:

```
Value of variable var is: 10
Value of variable var is: 10
Address of variable var is: 0x7fff5ed98c4c
Address of variable var is: 0x7fff5ed98c4c
Address of pointer p is: 0x7fff5ed98c50
```

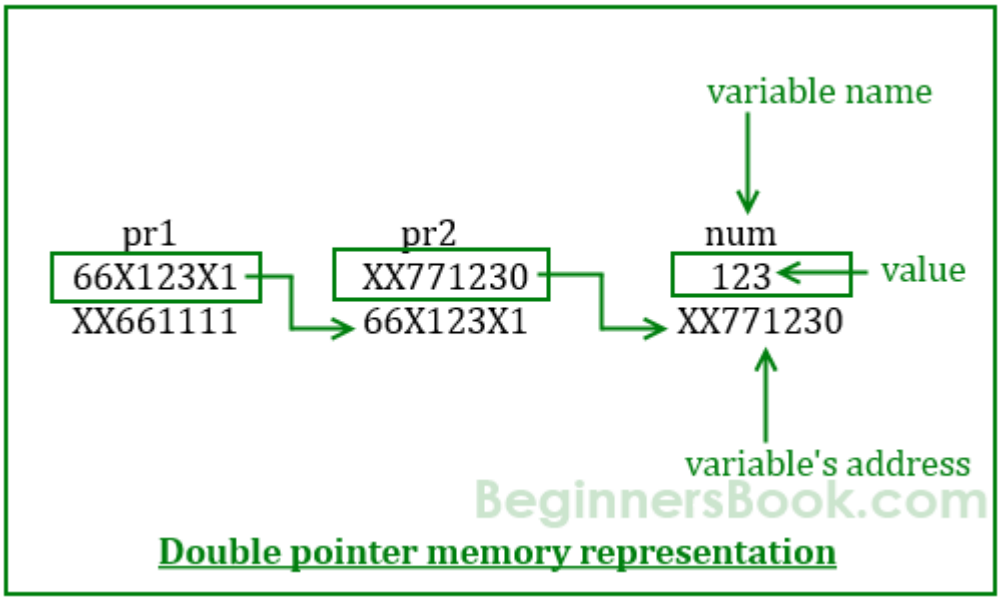


How to declare a Pointer to Pointer (Double Pointer) in C?

```
int **pr;
```

Here pr is a double pointer. There must be two *'s in the declaration of double pointer.

Let's understand the concept of double pointers with the help of a diagram:



As per the diagram, pr2 is a normal pointer that holds the address of an integer variable num. There is another pointer pr1 in the diagram that holds the address of another pointer pr2, the pointer pr1 here is a pointer-to-pointer (or double pointer).

Values from above diagram:

```
Variable num has address: XX771230
Address of Pointer pr1 is: XX661111
Address of Pointer pr2 is: 66X123X1
```

Passing Pointer to a Function in C Programming

In this example, we are passing a pointer to a function. When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C.

Swapping two numbers using Pointers

This is one of the most popular example that shows how to swap numbers using call by reference.

```
#include <stdio.h>
void swapnum(int *num1, int *num2)
{
```

```
   int tempnum;

   tempnum = *num1;
   *num1 = *num2;
   *num2 = tempnum;
}
int main( )
{
   int v1 = 11, v2 = 77 ;
   printf("Before swapping:");
   printf("\nValue of v1 is: %d", v1);
   printf("\nValue of v2 is: %d", v2);

   /*calling swap function*/
   swapnum( &v1, &v2 );

   printf("\nAfter swapping:");
   printf("\nValue of v1 is: %d", v1);
   printf("\nValue of v2 is: %d", v2);
}
```

Output:

```
Before swapping:
Value of v1 is: 11
Value of v2 is: 77
After swapping:
Value of v1 is: 77
Value of v2 is: 11
```

Try this program without pointers, you would see that the numbers are not swapped. This is because the change made by the function would be done to the local variables of the function. When we use pointers, the value is changed at the address of variable.

How to declare a function pointer?

```
function_return_type(*Pointer_name)(function argument list)
```

For example:

```
double  (*p2f)(double, char)
```

Here double is a return type of function, p2f is name of the function pointer and (double, char) is an argument list of this function. Which means the first argument of this function is of double type and the second argument is char type.

Lets understand this with the help of an example: Here we have a function sum that calculates the sum of two numbers and returns the sum. We have created a pointer f2p that points to this function, we are invoking the function using this function pointer f2p.

```
int sum (int num1, int num2)
{
   return num1+num2;
}
int main()
{

   /* The following two lines can also be written in a single
    * statement like this: void (*fun_ptr)(int) = &fun;
    */
   int (*f2p) (int, int);
   f2p = sum;
   //Calling function using function pointer
   int op1 = f2p(10, 13);

   //Calling function in normal way using function name
   int op2 = sum(10, 13);

   printf("Output1: Call using function pointer: %d",op1);
   printf("\nOutput2: Call using function name: %d", op2);

   return 0;
}
```

Output:

```
Output1: Call using function pointer: 23
Output2: Call using function name: 23
```

Some points regarding function pointer:

1. As mentioned in the comments, you can declare a function pointer and assign a function to it in a single statement like this:

```
void (*fun_ptr)(int) = &fun;
```

2. You can even remove the ampersand from this statement because a function name alone represents the function address. This means the above statement can also be written like this:

```
void (*fun_ptr)(int) = fun;
```

Pointer and Array in C programming with example

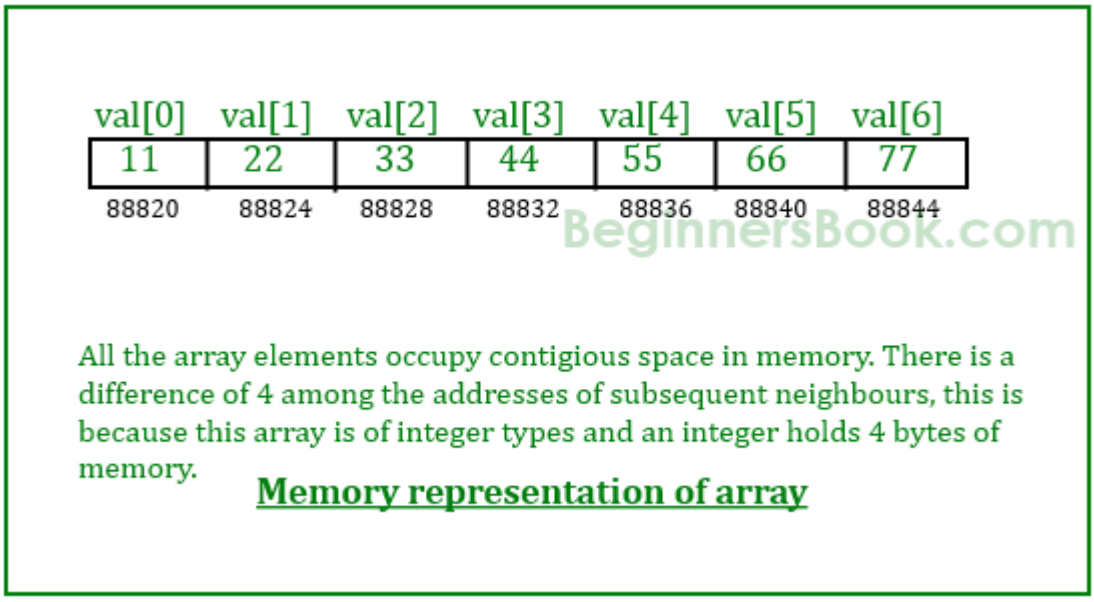
A simple example to print the address of array elements

```
#include <stdio.h>
int main( )
{
  int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;
  /* for loop to print value and address of each element of array*/
  for ( int i = 0 ; i < 7 ; i++ )
  {

    printf("val[%d]: value is %d and address is %d\n", i, val[i], &val[i]);
  }
  return 0;
}
```

Output:

```
val[0]: value is 11 and address is 1423453232
val[1]: value is 22 and address is 1423453236
val[2]: value is 33 and address is 1423453240
val[3]: value is 44 and address is 1423453244
val[4]: value is 55 and address is 1423453248
val[5]: value is 66 and address is 1423453252
val[6]: value is 77 and address is 1423453256
```

Note that there is a difference of 4 bytes between each element because that's the size of an integer. Which means all the elements are stored in consecutive contiguous memory locations in the memory.(See the diagram below)



In the above example I have used &val[i] to get the address of ith element of the array. We can also use a pointer variable instead of using the ampersand (&) to get the address.

Example – Array and Pointer Example in C

```
#include <stdio.h>
int main( )
{
  /*Pointer variable*/
  int *p;

  /*Array declaration*/
  int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;

  /* Assigning the address of val[0] the pointer
   * You can also write like this:
   * p = var;
   * because array name represents the address of the first element
   */
  p = &val[0];

  for ( int i = 0 ; i<7 ; i++ )
  {
    printf("val[%d]: value is %d and address is %p\n", i, *p, p);
    /* Incrementing the pointer so that it points to next element
     * on every increment.
     */
    p++;
  }
  return 0;
}
```

Output:

```
val[0]: value is 11 and address is 0x7fff51472c30
val[1]: value is 22 and address is 0x7fff51472c34
val[2]: value is 33 and address is 0x7fff51472c38
val[3]: value is 44 and address is 0x7fff51472c3c
val[4]: value is 55 and address is 0x7fff51472c40
val[5]: value is 66 and address is 0x7fff51472c44
val[6]: value is 77 and address is 0x7fff51472c48
```

Points to Note:

1) While using pointers with array, the data type of the pointer must match with the data type of the array.

2) You can also use array name to initialize the pointer like this:

```
p = var;
```

because the array name alone is equivalent to the base address of the array.

```
val==&val[0];
```

3) In the loop the increment operation(p++) is performed on the pointer variable to get the next location (next element's location), this arithmetic is same for all types of arrays (for all data types double, char, int etc.) even though the bytes consumed by each data type is different.

Pointer logic

You must have understood the logic in above code so now it's time to play with few pointer arithmetic and expressions.

```
if p = &val[0] which means
*p ==val[0]
(p+1) == &val[2]  & *(p+1) == val[2]
(p+2) == &val[3]  & *(p+2) == val[3]
(p+n) == &val[n+1) & *(p+n) == val[n+1]
```

Using this logic we can rewrite our code in a better way like this:

```
#include <stdio.h>
int main( )
{
   int *p;
   int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;
   p = val;
```

```
  for ( int i = 0 ; i<7 ; i++ )
  {
    printf("val[%d]: value is %d and address is %p\n", i, *(p+i), (p+i));
  }
  return 0;
}
```

We don't need the p++ statement in this program.

File Handling in C

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file

Functions for file handling

There are many functions in the C library to open, read, write, search and close the file. A list of file functions is given below:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

1. Opening a file

fopen() function is used for opening a file.

Syntax:

```
FILE pointer_name = fopen ("file_name", "Mode");
```

pointer_name can be anything of your choice.
file_name is the name of the file, which you want to open. Specify the full path here like

"C:\\myfiles\\newfile.txt".

While opening a file, you need to specify the mode. The mode that we use to read a file is "r" which is "read only mode".

for example:

```
FILE  *fp;
fp = fopen("C:\\myfiles\\newfile.txt", "r");
```

The address of the first character is stored in pointer fp.

How to check whether the file has opened successfully?
If file does not open successfully then the pointer will be assigned a NULL value, so you can write the logic like this:
This code will check whether the file has opened successfully or not. If the file does not open, this will display an error message to the user.

```
..
FILE fpr;
fpr = fopen("C:\\myfiles\\newfile.txt", "r");
if (fpr == NULL)
{
   puts("Error while opening file");
   exit();
}
```

Various File Opening Modes:

The file is opened using fopen() function, while opening you can use any of the following mode as per the requirement.

Mode "r": It is a read only mode, which means if the file is opened in r mode, it won't allow you to write and modify content of it. When fopen() opens a file successfully then it returns the address of first character of the file, otherwise it returns NULL.

Mode "w": It is a write only mode. The fopen() function creates a new file when the specified file doesn't exist and if it fails to open file then it returns NULL.

Mode "a": Using this mode Content can be appended at the end of an existing file. Like Mode "w", fopen() creates a new file if it file doesn't exist. On unsuccessful open it returns NULL.
File Pointer points to: last character of the file.

Mode "r+": This mode is same as mode "r"; however you can perform various operations on the file opened in this mode. You are allowed to read, write and modify the content of file opened in "r+" mode.
File Pointer points to: First character of the file.

Mode "w+": Same as mode "w" apart from operations, which can be performed; the file can be read, write and modified in this mode.

Mode "a+": Same as mode "a"; you can read and append the data in the file, however content modification is not allowed in this mode.

2. Reading a File

To read the file, we must open it first using any of the mode, for example if you only want to read the file then open it in "r" mode. Based on the mode selected during file opening, we are allowed to perform certain operations on the file.

C Program to read a file

fgetc( ): This function reads the character from current pointer's position and upon successful read moves the pointer to next character in the file. Once the pointers reaches to the end of the file, this function returns **EOF (End of File)**. We have used **EOF** in our program to determine the end of the file.

```
#include <stdio.h>
int main()
{
   /* Pointer to the file */
   FILE *fp1;
   /* Character variable to read the content of file */
   char c;

   /* Opening a file in r mode*/
   fp1= fopen ("C:\\myfiles\\newfile.txt", "r");
```

```
    /* Infinite loop –I have used break to come out of the loop*/
    while(1)
    {
       c = fgetc(fp1);
       if(c==EOF)
          break;
       else
          printf("%c", c);
    }
    fclose(fp1);
    return 0;
}
```

3. Writing to a file

To write the file, we must open the file in a mode that supports writing. For example, if you open a file in “r” mode, you won’t be able to write the file as “r” is read only mode that only allows reading.

Example: C Program to write the file

This program asks the user to enter a character and writes that character at the end of the file. If the file doesn’t exist then this program will create a file with the specified name and writes the input character into the file.

```
#include <stdio.h>
int main()
{
   char ch;
   FILE *fpw;
   fpw = fopen("C:\\newfile.txt","w");

   if(fpw == NULL)
   {
      printf("Error");
      exit(1);
   }

   printf("Enter any character: ");
   scanf("%c",&ch);

   /* You can also use fputc(ch, fpw);*/
   fprintf(fpw,"%c",ch);
   fclose(fpw);

   return 0;
}
```

4. Closing a file

```
fclose(fp);
```

The **fclose( )** function is used for closing an opened file. As an argument you must provide a pointer to the file that you want to close.

An example to show Open, read, write and close operation in C

```
#include <stdio.h>
int main()
{
   char ch;

   /* Pointer for both the file*/
   FILE *fpr, *fpw;
   /* Opening file FILE1.C in “r” mode for reading */
   fpr = fopen("C:\\file1.txt", "r");

   /* Ensure FILE1.C opened successfully*/
   if (fpr == NULL)
   {
       puts("Input file cannot be opened");
   }

   /* Opening file FILE2.C in “w” mode for writing*/
   fpw= fopen("C:\\file2.txt", "w");

   /* Ensure FILE2.C opened successfully*/
   if (fpw == NULL)
   {
     puts("Output file cannot be opened");
   }

   /*Read & Write Logic*/
   while(1)
   {
      ch = fgetc(fpr);
      if (ch==EOF)
         break;
      else
         fputc(ch, fpw);
   }

   /* Closing both the files */
   fclose(fpr);
   fclose(fpw);

   return 0;
}
```

Program to write odd and even numbers to separate files

```
#include <stdio.h>
int main()
{
  FILE *f1, *f2, *f3;
  int number, i;
  printf("Contents of DATA file\n\n");
  f1 = fopen("DATA", "w"); /* Create DATA file */
  for(i = 1; i <= 30; i++)
  {
    scanf("%d", &number);
    if(number == -1) break;
    putw(number,f1);
  }
fclose(f1);
f1 = fopen("DATA", "r");
f2 = fopen("ODD", "w");
f3 = fopen("EVEN", "w");
/* Read from DATA file */
while((number = getw(f1)) != EOF)
{
   if(number %2 == 0)
     putw(number, f3); /* Write to EVEN file */
   else
     putw(number, f2); /* Write to ODD file */
}
fclose(f1);
fclose(f2);
fclose(f3);
f2 = fopen("ODD","r");
f3 = fopen("EVEN", "r");
printf("\n\nContents of ODD file\n\n");
while((number = getw(f2)) != EOF)
    printf("%4d", number);
printf("\n\nContents of EVEN file\n\n");
while((number = getw(f3)) != EOF)
    printf("%4d", number);
fclose(f2);
fclose(f3);
return 0;
}
```

puts takes two arguments –

```
fputs(str, fpw)
```

str – str represents the array, in which string is stored.
fpw – FILE pointer to the output file, in which record needs to be written.

Point to note about fputs:

fputs by default doesn’t add new line after writing each record, in order to do that manually – you can have the following statement after each write to the file.

```
fputs("\n", fpw);
```

Dynamic memory allocation in C

The concept of **dynamic memory allocation in c language** *enables the C programmer to allocate memory at runtime*. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Before learning above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

static memory allocation	dynamic memory allocation
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Now let's have a quick look at the methods used for dynamic memory allocation.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

malloc() function in C

The malloc() function allocates single block of requested memory.

It doesn't initialize memory at execution time, so it has garbage value initially.

It returns NULL if memory is not sufficient.

The syntax of malloc() function is given below:

1. ptr=(cast-type*)malloc(byte-size)

Let's see the example of malloc() function.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int n,i,*ptr,sum=0;
printf("Enter number of elements: ");
scanf("%d",&n);
ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
 if(ptr==NULL)
{
          printf("Sorry! unable to allocate memory");
           exit(0);
}
printf("Enter elements of array: ");
for(i=0;i<n;++i)
{
       scanf("%d",ptr+i);
       sum+=*(ptr+i);
}
printf("Sum=%d",sum);
free(ptr);
return 0;

}
```

Output

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

calloc() function in C

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

1. ptr=(cast-type*)calloc(number, byte-size)

Let's see the example of calloc() function.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));  //memory allocated using calloc
    if(ptr==NULL)
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

Output

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

Let's see the syntax of realloc() function.

1. ptr=realloc(ptr, **new**-size)

free() function in C

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

Let's see the syntax of free() function.

1. free(ptr)