1. Pointers

Pointer is a variable that stores a memory address. Every variable is located under unique location within a computer's memory and this unique location has its own unique address, the memory address. Normally, variables hold values such as 5 or "hello" and these values are stored under specific location within computer memory. However, **pointer is a different beast, because it holds the memory address as its value and has an ability to "point" (hence pointer) to certain value within a memory, by use of its associated memory address.**

1.1 Retrieving a Variable's Memory Address

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

    int myInt = 99; // Declare an integer variable and initialize it with 99
     printf(" value is: %d ",myInt); // Print out value of myInt

    // Use address-of operator & to print out a memory address of myInt

    printf("\n Address of myInt is: %d ",&myInt);

    return 0;
    }
```

Assigning a Variable's Memory Address to a Pointer

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

int * pPointer = 0; // Declare and initialize a pointer.
int twoInt = 35698; // Declare an integer variable and initialize it with 35698
int oneInt = 77; // Declare an integer variable and initialize it with 77
// Use address-of operator & to assign a memory address of twoInt to a pointer
pPointer = &twoInt;
// Pointer pPointer now holds a memory address of twoInt

// Print out associated memory addresses and its values
printf("pPointer's memory address:%d \n",&pPointer);
printf("\nInteger's oneInt memory address:%d \n", &oneInt);
printf("Integer value:%d \n",oneInt);
printf("\nInteger's twoInt memory address:%d\n",&twoInt);
printf("Integer value:%d\n" ,twoInt);
printf("\npPointer is pointing to memory address:%d\n",pPointer);
printf("Integer value:%d" , *pPointer);

return 0;
}
```

Initializing the Pointer via the Address-Of Operator (&)

When you declare a pointer variable, its content is not initialized. In other words, it contains an address of "somewhere", which is of course not a valid location. This is dangerous! You need to initialize a pointer by assigning it a valid address. This is normally done via the **address- of operator (&).**

The **address-of operator (&)** operates on a variable, and returns the address of the variable. For example, if number is an int variable, &number returns the address of the variable number.

You can use the address-of operator to get the address of a variable, and assign the address to a pointer variable.

For example,

```
int number = 88; // An int variable with a value
int * pNumber; // Declare a pointer variable called pNumber pointing to an int (or int pointer)
pNumber = &number;// Assign the address of the variable number to pointer pNumber
int * pAnother = &number // Declare another int pointer and init to address of the variable numbe
```

Accessing the Value at the Memory Address held by a Pointer

```
#include<stdio.h>
int main()
{
int myInt = 99; // Declare an integer variable and initialize it with 99
int * pMark = 0;// Declare and initialize a pointer
printf("value of myInt is %d\n", myInt); // Print out a value of myInt
pMark = &myInt;// Use address-of operator & to assign a memory address of myInt to a pointer
printf("Here is the value of *pMark %d\n", *pMark ); // Dereference a pMark pointer with dereference operator * to access a value of myInt

return 0;
}
```

Manipulating Data with Pointers

```
#include<stdio.h>
int main()
{
int myInt = 99; // Declare an integer variable and initialize it with 99
int * pMark = 0;// Declare and initialize a pointer
printf("value of myInt is %d\n", myInt); // Print out a value of myInt
pMark = &myInt;// Use address-of operator & to assign a memory
address of myInt to a pointer
// dereference a pMark pointer with dereference operator * and set
new value
*pMark = 11;
// show indirectly a value of pMark and directly the value of myInt
printf("*pMark:%d\t\n", *pMark );
printf("\nmyInt:%d\t", myInt );

return 0;
}
```

Pointers has a type Too

A pointer is associated with a type (of the value it points to), which is specified during declaration. A pointer can only hold an address of the declared type; it cannot hold an address of a different type.

```
int i = 88;
double d = 55.66;
int * iPtr = &i;    // int pointer pointing to an int value
double * dPtr = &d; // double pointer pointing to a double value

iPtr = &d;   // ERROR, cannot hold address of different
type dPtr = &i;      // ERROR
iPtr = i;    // ERROR, pointer holds address of an int, NOT int value

int j = 99;
iPtr = &j; // You can change the address stored in a pointer
```

Test Example of Pointers

```
#include<stdio.h> int
main()
{
int number = 88; // Declare an int variable and assign an initial value
int * pNumber;// Declare a pointer variable pointing to an int (or int
                pointer)
pNumber = &number; // assign the address of the variable number to
                pointer pNumber

printf("pNumber holds the content: %d\n",pNumber); // Print content
                                                    of pNumber
printf("address of number is %d \n", &number); // Print address of
                                                    number
printf("value pointed to by pNumber is %d: \n",*pNumber);// Print
value pointed to by pNumber
printf("value of number is %d: \n",number); // Print value of number (88)
*pNumber = 99;              // Re-assign value pointed to by pNumber
printf("content of pNumber: %d\n ",pNumber);// Print content of pNumber
printf("Address of number is :%d ",&number); // Print address of number
printf("value pointed to by pNumber %d: \n",*pNumber); // Print
value pointed to by pNumber (99)
printf("Now value od number is %d \n", number );// Print value of number
(99)
// The value of number changes via pointer
printf(" the address of pointer variable pNumber is %d: ",&pNumber);
// Print the address of pointer variable pNumber return 0;
}
```

Passing pointers to functions in C

C allows you to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type.

```
#include <stdio.h>
//#include<stdlib.h>
double CalculateNetPrice(double *);

int main()
{
 double FinalPrice;
 double Discount = 20.00;

 FinalPrice = CalculateNetPrice(&Discount);

 printf("After applying Discount\n");
 printf("Final Price =
 %d\n",FinalPrice);

 return 0;
}

double CalculateNetPrice(double *Discount)
{
 double OrigPrice;

 printf("Enter the original price: ");
 scanf("%d",&OrigPrice);

 return OrigPrice - (OrigPrice * *Discount / 100);
}
```

Array and Pointer variables

The function which can accept a pointer, can also accept an array

```
#include <stdio.h>
//#include<stdlib.h>
double getAverage(int *arr, int size); // function declaration:
int main ()
{
int balance[5] = {1000, 2, 3, 17, 50}; // an int array with 5 elements.
double avg;
 avg = getAverage( balance, 5 ) ;// pass pointer to the array as an
                                    argument.

printf("Average value is: %g ", avg); // output the returned value
return 0;
}

double getAverage(int *arr, int size)
{
int     i, sum = 0; double avg;
for (i = 0; i < size; ++i)
        {
                sum += arr[i];
        }
avg = double(sum) / size; return avg;
}
```

C Dynamic Memory Allocation: **malloc(),free()**

The exact size of array is unknown untill the compile time,i.e., time when a compier compiles code written in a programming language into a executable form. The size of array you have declared initially can be sometimes insufficient and sometimes more than required. Dynamic memory allocation allows a program to obtain more memory space, w hile running or to release space when no space is required.

Although, C language inherently does not has any technique to allocated memory dynamically, there are some library functions under "stdlib.h" for dynamic memory allocation.

Function	Use of Function
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
free()	dellocate the previously allocated space

malloc()

The name malloc stands for "memory allocation". The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.

Syntax of malloc()

```
ptr=(cast-type*)malloc(byte-size)
```

Here, ptr is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

```
ptr=(int*)malloc(100*sizeof(int));
```

This statement will allocate either 200 or 400 according to size of int 2 or 4 bytes respectively and the pointer points to the address of fir st byte of memory.

Example of malloc()

A program to find sum of n elements entered by user.

```
#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("Error! memory not allocated."); 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;
}
```

free()

Dynamically allocated memory with malloc() does not get return on its own. The programmer must use free() explicitly to release space.

syntax of free()

```
free(ptr);
```

This statement cause the space in memory pointer by ptr to be deallocated.

```
#include <stdio.h>

#include <stdlib.h>

int main (void)

{
  int * p;
  p = (int *) malloc  (sizeof (int));       /* partitioning of type int */

  if (p == NULL)// failed to reserve area *  /

  {
  printf ("Failed to allocate space for% d bytes", sizeof (int));

  return 1;

  }
  * p =  150;
  printf ("% d ", *  p);

  free (p); // free area

  * / return 0;

}
```

Return an Array from a Function

```
#include <stdio.h>
#include <conio.h>
        int* getEvenNumbers(int N)   /* This function returns an array of N
        even numbers */
        {
        static int evenNumberArray[100];          /* Declaration of a static local
        integer array */

          int i, even = 2;

            for(i=0; i<N; i++){
                evenNumberArray[i] = even;
                even += 2;
            }

            return evenNumberArray;    /* Returning base address of
        evenNumberArray array*/
        }

        int main(){
           int *array, counter;
           array = getEvenNumbers(10);
           printf("Even Numbers\n");
           for(counter=0; counter<10; counter++){
              printf("%d\n", array[counter]);
           }

           return 0;
        }
```

Another Example in 1D Array

```
#include<stdio.h>
#include<stdlib.h>
static int a[]={1,2};
int *fun()
{

        printf("%u\n",a);
        printf("%u\n",(a+1));
         return a;
}

int main()
```

```
{

  int *q=(int *)malloc(sizeof(int));
  int i=0;
        int *p=fun();
        printf("%u\n",p);
        printf("%u\n",p+1);
        while(i<2)
        {
              printf(" %d ",*p);
              p++;
              i++;

        }
        return 0;
}
```

Another Example in 2D Array

```
#include<stdio.h>
#include<stdlib.h>
int **fun()
{
int **a=(int**)malloc(2*sizeof(int));
for(int i=0;i<2;i++)
a[i]=(int*)malloc(2*sizeof(int));
 //a[2][2]={1,2,3,4};
 a[0][0]=1;
 a[0][1]=2;
 a[1][0]=3;
 a[1][1]=4;
 return a;
}

int main()
{
int i=0,j=0;
       int **p=fun();
       while(i<2)
       {
             for(j=0;j<2;j++)
                   printf(" %d ",p[i][j]);
```

```
        i++;
    }
    return 0;
}
```

Void Pointers in C : Definition

1. Suppose we have to declare integer pointer,character pointer and float pointer then we need to declare 3 pointer variables.
2. Instead of declaring different types of pointer variable it is feasible to declare single pointer variable which can act as integer pointer,character pointer.

Void Pointer Basics:

1. In C **General Purpose Pointer** is called as void Pointer.
2. It does not have any data type associated with it
3. It can store address of any type of variable
4. A void pointer is a C convention for a raw address.
5. The compiler has no idea what type of object a void Pointer really points to?

Declaration of Void Pointer:

```
void * pointer_name;
```

Void Pointer Example :

```
void *ptr;    // ptr is declared as Void pointer

char cnum;
int inum;
float fnum;

ptr = &cnum; // ptr has address of character data
ptr = &inum;  // ptr has address of integer data
ptr = &fnum; // ptr has address of float data
```

Explanation :

```
void *ptr;
```

1. **Void pointer** declaration is shown above.
2. We have declared 3 variables of integer,character and float type.
3. When we assign **address of integer** to the void pointer, pointer will become Integer Pointer.
4. When we assign **address of Character** Data type to void pointer it will become Character Pointer.
5. Similarly we can assign address of any data type to the void pointer.
6. It is capable of storing address of any data type