Problem Solving With C

UE15CS151

By:

Amit V Pujari

Asst. Professor

Department of Computer Science And Engineering

PES University

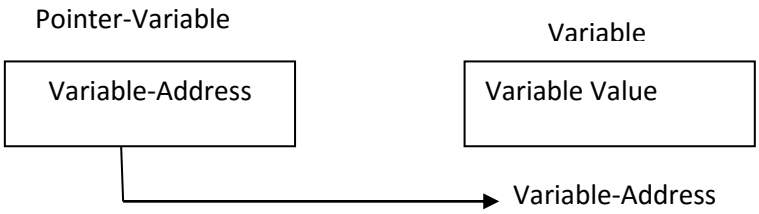
Mail : amith.v.pujari@gmail.com

POINTERS

This Notes can be used only for reference and kindly do not solely depend on it. Only those topics which need more explanation are included here. Please Note "The prescribed Text book has to be referred for the examination"

Pointers

In C every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory



Example :

```
int a=100;
printf("The address of a is %d",&a);
```

[The address is a hexadecimal value hence %x-[hexadecimal value] can also be used as the format specifier]

Output : 234532632 // Memory address of "a"

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address.

The general form of a pointer variable declaration is :

Type *variable_name;

Here, **type** is the pointer's base type; it must be a valid C data type and **variable_name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer

Example:

```
int *a;          //Pointer to Integer
float *f;        //Pointer to a Float
char *c          //Pointer to a Char
double *d;       //Pointer to a Double
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to

Example:

```
int *a;
float *f;
char *c
double *d;

printf("The size of pointer a is %d \t",sizeof(a));
printf("The size of pointer f is %d \t",sizeof(f));
printf("The size of pointer c is %d \t",sizeof(c));
printf("The size of pointer c is %d \t",sizeof(d));
```

/*sizeof() – is a builtin function that returns the size of a particular variable in terms of Bytes

Example :

```
Int a= 10;
Printf("%d",size(a));
```

Output: 4 // integer takes 4 Bytes but this varies with respect to the system */

Output : 8 8 8 8 //Size assigned to all the pointer variable is same .But the size depends on the system.

There are a few important operations, which we will do with the help of pointers very frequently.

(a) We define a pointer variable

(b) Assign the address of a variable to a pointer.

(c) Finally access the value at the address available in the pointer variable. This is done by using unary operator ***** that returns the value of the variable located at the address specified by its operand. Here '*' is referred as Dereferencing operator.

Example :

```
int a = 10;
int *i;
i = &a;
printf("The value of the variable is %d \n",a);
printf("The value of the address of the variable is %d \n",i);
```

printf("The value of the variable through the pointer is %d \n",*i);

Output:

The value of the variable is 10

The value of the address of the variable is 63542617

The value of the variable through the pointer is 10

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

Example :

int *p = NULL:

POINTER ARITHMETIC :

A pointer in c is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers: "++"," –", " +", and" –".

Increment: Assume the of "a" to be 1000

```
int a=100;
int *b;        //Pointer Variable
int c;         //Just a Variable
b = &a;
c = &a;
printf("%d  %d  \n", b ,c);
b++;
c++;
printf("%d  %d  \n", b ,c);
```

Output: 1000 1000
1004 1000 //Considering "int" to be of 4 Bytes

Decrement: Assume the address of "a" to be 1000

```
int a=100;
int *b;        //Pointer Variable
int c;         //Just a Variable
b = &a;
c = &a;
printf("%d  %d  \n", b ,c);
b--;
c--;
printf("%d  %d  \n", b ,c);
```

Output : 1000 1000
996 999 //Considering "int" to be of 4 Bytes

Pre and Post increment:

Example1 : Assuming the address of “a” to be 1000

```
int *p,a=100;
p = &a;
printf(“ Post increment value %d \n”, p++);
```

Output:

Post increment value 1000

//Here the assignment happens first and then the increment operation is performed hence the value still remains the same

Example2 : Assuming the address of “a” to be 1000

```
int *p,a=100;
p = &a;
printf(“ Pre increment value %d \n”, p++);
```

Output:

Pre increment value 1004

//Here the increment operation is done first and then assignment happens hence the result will be 1004

Example3: Assuming the address of “a” to be 1000

```
int *p,a=100;
p = &a;
printf(“ Post increment value %d \n”, p++);
printf(“ Pre increment value %d \n”, p++);
```

Output:

Post increment value 1000

Pre increment value 1008

//Pointer Decrement also works in the same way

Pointer Addition:

Example 1: Assuming the address of “a” to be 1000

```
int a,*p;
p = &a;
printf(“%d   \n”, p);
p = p +2;
printf(“%d   \n”, p);
```

Output :

```
1000
1008          // 1000+ (2*4[int size]) = 1008
```

//Addition of pointer to a constant value works

Example 2: Assuming the address of “a” to be 1000

```
int a,*p,y = 10;
p = &a;
printf(“%d   \n”, p);
p = p +y;
printf(“%d   \n”, p);
```

Output :

```
1000
1040          //1000 + (10*4) = 1040
```

//Addition of pointer to a variable works

Example 3: Assuming the address of “a” to be 1000 ,”x” is 2000

```
int a,*p,y = 10;
int *q ,x= 200
p = &a;
q=&x
printf(“%d   %d \n”, p,q);
p = p +q;
printf(“%d   %d   \n”, p,q );
```

Output :

```
ERROR
```

//Addition of 2 pointers doesn’t work

Pointer Subtraction :

Example 1: Assuming the address of “a” to be 1000

```
int a,*p;
p = &a;
printf(“%d  \n”, p);
p = p - 2;
printf(“%d  \n”, p);
```

Output :

```
1000
992          // 1000 - (2*4[int size]) = 992
```

//Addition of pointer to a constant value works

Example 2: Assuming the address of “a” to be 1000

```
int a,*p,y = 10;
p = &a;
printf(“%d  \n”, p);
p = p - y;
printf(“%d  \n”, p);
```

Output :

```
1000
960   //1000 - (10*4) = 1040
```

//Addition of pointer to a variable works

Example 3: Assuming the address of “a” to be 1000 ,”x” is 2000

```
int a,*p,y = 10;
int *q ,x= 200
p = &a;
q=&x
printf(“%d  %d \n”, p,q);
p = P - q;
printf(“%d  %d  \n”, p,q );
```

Output :

```
1000          2000
4152463725 2000
```

//Subtraction of 2 pointers works

Pointer to an Array

Before we study the concept of pointer to an array , we need to understand what the array identifier can do

Example 1 :

```
int a[5] = {11,22,33,44,55};
printf("%d \n",a);
```

Output:

624163578 // "a" stores the base address of the array is it has the memory address of the value at 0^{th} index of the array

Example 2:

```
Int a[5] ={11,22,33,44,55};
Printf("%d \n",a);
Printf("%d \n",*a);
```

Output :

624163578

11 // Value at the index 0

Here the Array Identifier acts like a pointer , but i will not be able to make changes to this vale as it is constant . Hence we Use the concept of Pointer to an Array.

Example 3:

```
int b[5] = {11,22,33,44,55};
for(i = 0;i<5;i++)
        printf("%d \t",b[i]);
```

output :

11 22 33 44 55

//This is the normal way of printing the array contents

Example 3:

```
Int *a;
Int b[5] = {11,22,33,44,55};
a = &b;
for(i = 0;i<5;i++,a++)
        printf("%d \t",*a);
```

output :

11 22 33 44 55

Array of Pointers :

As the name itself suggest we will create an array of pointers to store the address of another array values

Example :

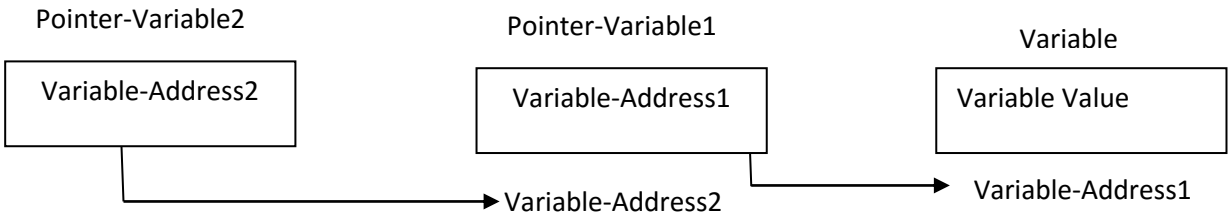
```
int *a[5];              // This is the declaration of Array of pointers
int b[5] = {1,2,3,4,5};
for(i=0;i<5;i++)
        a[i] = &b[i];
for(i=0;i<5;i++)
        printf("%d  :  %d\n", b[i],a[i],*a[i]);
```

output :

```
1       1000        1
2       1004        2
3       1008        3
4       1012        4
5       1016        5
```

Pointer To a Pointer:

Pointer to a pointer is a form of multiple indirections, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example

Type **Var;

Example :

```
int *a;          // “a” is a Pointer to a Variable

int **b;         //”b” Pointer to a Pointer

int c= 200;

a= &c;           // “a” is pointing to c

b = &a;          //”b” is a pointer to a pointer “a”

printf(“%d   %d  %d ”,c,*a,**b );
```

output:

```
200   200   200
```