Pointers In C

Pointers:

Pointer is a variable contain the address of another variable. If a variable contains address of another variable than it is said that first variable points to second. All operation perform on pointers are done through two operators '*' and '&'. '&' is a unary operator that returns a memory address of a variable. '*' is complement of '&' and return value stored at a memory location stored in a pointer. '*' can interpreted as statement "at address" while '&' can be interpreted as statement "address of".

Pointer Declaration:

Declaring a pointer variable is quite similar to declaring a normal variable all you have to do is to insert a star '*' operator before it.General form of pointer declaration is -

datatype* name;

where datatype represent the type of data to which pointer thinks it is pointing to.

Multiple pointers of similar type can be declared in one statement but make sure you use * before every one otherwise they will become a variable of that type.

Example:

```
 int *p;
 float *f1,*f2;
 char *ch;
```

Pointer Assignment:

Once we declare a pointer variable we must point it to a value by assigning the address of the variable

Example:

```
 int *p;
 int x;
 p=&x;
```

The value of one pointer can be asssigned to another pointer using assignment operator '=' . In this value of right hand side points to memory address of variable stored in left hand side pointer. As a result both pointers point to same memory location after this expression. Pointer of similar type can be used in expression easily as shown below but for diffrent type pointers you need to type cast them as shown in next section.

```
#include < stdio.h >
int main ()
{
char ch = 'x';
char *c1, *c2;
c1 = &ch;
c2 = c1; // Pointer Assignement Taking Place
printf (" *c1 = %c And *c2 = %c", *c1,*c2); // Prints 'x' twice
return 0;
}
```

Pointer Conversion

Before concept of pointer conversion you must understand the concept of a void pointer. Void pointer technically is a pointer which is pointing to the unknown. Void pointer has special property that it can be type casted into anyother pointer without any type casting though every other conversion needs an type casting. In dynamic memory allocation function such as malloc ( ) and calloc ( ) returns void pointer which can be easily converted to other types.

Also there is a pointer called null pointer which seems like void pointer but is entirely diffrent. Null pointer is a pointer which points to nothing. Null pointer points to the base address of the CPU register and since register is not addressable usage of a null pointer will lead to crash or at minimum a segmentation fault.

Also be careful while typecasting one pointer to another because even after type casting your pointer can point to anything but it will still think it is pointing to something of it declared type and have properties of the orignal type.

Type conversion is a powerful feature but yet it may lead difficult to remove bugs and crashes,it may also lead to unexpected and unreliable results but program would compile succesfully.

Code below shows a type casting of one pointer into another -

```
#include < stdio.h >
int main ()
{
int x=10;
char *ch;
int *p;
p = &x;
ch = (char *) p; // Type Casting and Pointer Conversion
printf (" *ch = %c And *p = %d", *ch,*p); // Output maybe unexpected depending on the
compiler.
return 0;
}
```

Pointer Expressions:

Like normal variables pointer variable can be used in expressions.

Example:consider 2 integer pointers p1,p2

```
int *p1,*p2;
int x,y,z,k;
p1=&x;
p2=&y;
z=*p1**p2;
k=k+*p1;
z=10**p2/*p1;
```

Pointer Arithmetic

Pointer arithemetic is quite diffrent from normal arithemetic. Not all artihemetic operations are defined in pointers. You can increment them, decrement them, add and subtract integer values from them. You even can subtract two pointers.But you cannot add two pointers, mulitply, divide,modulus them. You can not also add or subtract values other than integer.

Address+Number=Address
Address-Number=Address
Address++=Address
Address--=Address

Now consider a pointer X , its current value that address it is pointing to is 1000 (just assuming).We make another assumption about the size of the data types. Size of data type is machine dependent, for example int can be 2,4 byte depending upon the compiler.
Now if this X pointer is char type(assumed 1 Byte) then X++ will have value 1001 and X-- will have value 999. Now if this X pointer is integer type (assumed 2 byte) then X++ will have value 1002 and X-- will have value 998. Again if this X pointer is float type (assumed 4 Byte) than X++ will have value 1004 and X-- will have value 996. Also if this X pointer is double type(assumed 8 Byte) than X++ will have value 1008 and X-- will have value 992.

when you increment a pointer of certain base type it increase it value in such a way that it points to next element of its base type. If you decrement a pointer its value decrease in such a way that it points to previous value of its base type.

You can add or subtract any integer value, in such case value of pointer get increase and decrease by the product of the value to be added or subtract and size of the base type. Pointer of user defined types such as structures and union also increase by the quantity of thier bit values which can be determined using sizeof operator.

Pointer Comparison:

Two pointers can be compared no matter where they point. Comparison can be done using <, >, =, <= and >= operators. Though it is not forcibly implied but comparison of two

pointers become sensible only when they are related such as when they are pointing to element of same arrays.Comparison of two unrelated pointers is unpredictable and your code should not rely upon it. All comparison are generally done on basis of memory organization in the host machine.
Following C source code shows pointer comparison in C -

```
#include < stdio.h >
int main ()
{
int data[10],i;
int* p1,*p2;
for (i = 0; i <10;i++)
{
data[i] = i;
}
p1 = &data [1];
p2 = &data [2];
if (p1 > p2)
{
printf ("p1 is greater than p2\n");
}
else
{
printf ("p2 is greater than p1\n");
}
}
output:p2 is greater than p1
```

Source:

http://datastructuresprogramming.blogspot.in/2010/05/pointers-pointer-is-variable-contain.html