Lecture 1

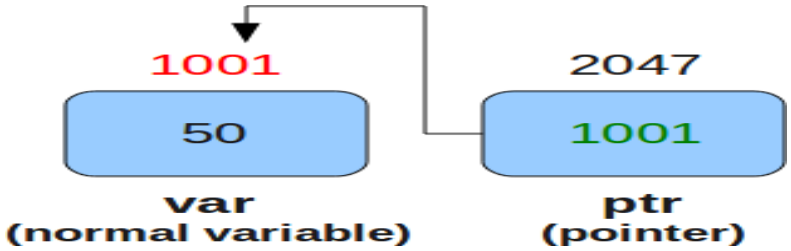
Pointer

What are Pointers?

Different from other normal variables which can store values.

pointers are special variables that can hold the address of a variable. Since they store memory address of a variable, the pointers are very commonly said to “point to variables”.

Lets try to understand the concept.



As shown in the above diagram:

- A normal variable ‘var’ has a memory address of 1001 and holds a value 50.
- A **pointer variable** has its own address 2047 but stores 1001, which is the address of the variable ‘var’

A pointer is declared as :

<pointer type> *<pointer-name>

In the above declaration :

pointer-type : It specifies the type of pointer. It can be int,char, float etc. This type specifies the type of variable whose address this pointer can store.

pointer-name : It can be any name specified by the user. Professionally, there are some coding styles which every code follows. The pointer names commonly start with ‘p’ or end with ‘ptr’

An example of a pointer declaration can be :

```
char *chptr;
```

In the above declaration,

- ‘char’ signifies the pointer type,
- chptr is the name of the pointer
- while the asterisk ‘*’ signifies that ‘chptr’ is a pointer variable.

A pointer is initialized in the following way :

<pointer declaration(except semicolon)> = <address of a variable>

OR

<pointer declaration> <name-of-pointer> = <address of a variable>

Note that the type of variable above should be same as the pointer type.(Though this is not a strict rule but for beginners this should be kept in mind).

For example :

```
char ch = 'c';
char *chptr = &ch; //initialize
```

OR

```
char ch = 'c';
char *chptr; chptr = &ch //initialize
```

In the code above, we declared a character variable ch which stores the value ‘c’. Now, we declared a character pointer ‘chptr’ and initialized it with the address of variable ‘ch’.

Note that the ‘&’ operator is used to access the address of any type of variable.

A pointer can be used in two contexts.

Context 1: For accessing the address of the variable whose memory address the pointer stores.

Again consider the following code :

```
char ch = 'c';
char *chptr = &ch;
```

Now, whenever we refer the name ‘chptr’ in the code after the above two lines, then compiler would try to fetch the value contained by this pointer variable, which is the address of the variable (ch) to which the pointer points. i.e. the value given by ‘chptr’ would be equal to ‘&ch’.

For example :

```
char *ptr = chptr;
```

The value held by ‘chptr’ (which in this case is the address of the variable ‘ch’) is assigned to the new pointer ‘ptr’.

Context 2: For accessing the value of the variable whose memory address the pointer stores.

Continuing with the piece of code used above :

```
char ch = 'c';
char t;
char *chptr = &ch;
t = *chptr;
```

We see that in the last line above, we have used '*' before the name of the pointer. What does this asterisk operator do?

Well, this operator when applied to a pointer variable name(like in the last line above) yields the value of the variable to which this pointer points. Which means, in this case '*chptr' would yield the value kept at address held by chptr. Since 'chptr' holds the address of variable 'ch' and value of 'ch' is 'c', so '*chptr' yeilds 'c'.

Consider the following code :

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


 {
char ch = 'c';
char *chptr = &ch;

 int i = 20;
int *intptr = &i;

float f = 1.20000;
 float *fptr = &f;

char *ptr = "I am a string";

 printf("\n [%c], [%d], [%f], [%c], [%s]\n", *chptr, *intptr, *fptr, *ptr, ptr);
return 0;
}
```

OUTPUT :

```
$ ./pointers
[c], [20], [1.200000], [I], [I am a string]
```

Consider the following code :

```
#include<stdio.h>
struct st
{ int a;
char ch;
 };
int main(void)
 {
          struct st obj;
         struct st *stobj = &obj;

          stobj->a = 5;
          stobj->ch = 'a';

          printf("\n [%d] [%c]\n", stobj->a, stobj->ch);
         return 0;
}
```

OUTPUT:
[5] [a]

In the above code, we have declared a pointer stobj of type ‘struct st’. Now since the pointer type is a structure, so the address it points to has to be of a ‘struct st’ type variable(which in this case is ‘obj’).

Other interesting part is how structure elements are accessed using pointer variable ‘stobj’. Yes, When dealing with pointer objects, its a standard to use arrow operator -> instead of ‘.’ operator(which would have been used, had we used ‘obj’ to access the structure elements).