**Introduction to Pointers**

**Pointers are variables that hold address of another variable of same data type**.

Pointers are one of the most distinct and exciting features of C language. It provides power and flexibility to the language. Its a powerful tool and handy to use once its mastered.

**Benefit of using pointers**

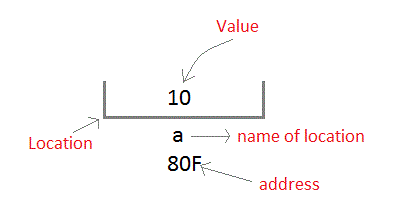
* Pointers are more efficient in handling Array and Structure.
* Pointer allows references to function and thereby helps in passing of function as arguments to other function.
* It reduces length and the program execution time.
* It allows C to support dynamic memory management.

**Concept of Pointer**

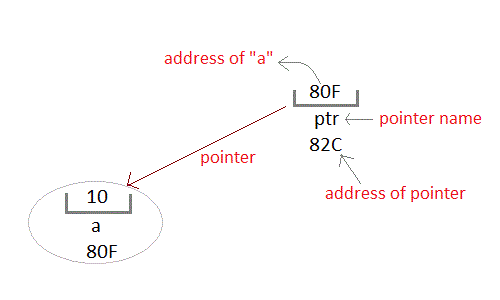
Whenever a **variable** is declared, system will allocate a location to that variable in the memory, to hold value. This location will have its own address number.

Let us assume that system has allocated memory location 80F for a variable **a**.

int a = 10 ;



**We can access the value 10 by either using the variable name a or the address 80F**. Since the memory addresses are simply numbers they can be assigned to some other variable. The variable that holds memory address are called **pointer variables**. A **pointer** variable is therefore nothing but a variable that contains an address, which is a location of another variable. Value of **pointer variable** will be stored in another memory location.



### Declaring a pointer variable

General syntax of pointer declaration is,

*data-type \*pointer\_name;*

Data type of

pointer must be same as the variable, which the pointer is pointing. **void** type pointer works with all data types, but isn't used oftenly.

#### Initialization of Pointer variable

**Pointer Initialization** is the process of assigning address of a variable to **pointer** variable. Pointer variable contains address of variable of same data type. In C language **address operator** **&** is used to determine the address of a variable. **The & (**immediately preceding a variable name) returns the address of the variable associated with it.

int a = 10 ;

int \*ptr ; *//pointer declaration*

ptr = &a ; *//pointer initialization*

or,

int \*ptr = &a ; *//initialization and declaration together*

Pointer variable always points to same type of data.

float a;

int \*ptr;

ptr = &a; //ERROR, type mismatch

#### Dereferencing of Pointer

Once a pointer has been assigned the address of a variable. To access the value of variable, pointer is dereferenced, using the **indirection operator** \*.

int a,\*p;

a = 10;

p = &a;

printf("%d",\*p); *//this will print the value of a.*

printf("%d",\*&a); *//this will also print the value of a.*

printf("%u",&a); *//this will print the address of a.*

printf("%u",p); *//this will also print the address of a.*

printf("%u",&p); *//this will also print the address of p.*

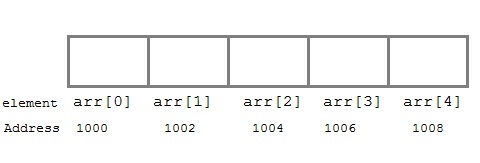
### Pointer and Arrays

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address which gives location of the first element is also allocated by the compiler.

Suppose we declare an array **arr**,

int arr[5]={ 1, 2, 3, 4, 5 };

Assuming that the base address of **arr** is 1000 and each integer requires two byte, the five element will be stored as follows



Here variable **arr** will give the base address, which is a constant pointer pointing to the element, **arr[0]**. Therefore **arr** is containing the address of **arr[0]** i.e 1000.

**arr** *is equal to* **&arr[0]** // by default

We can declare a pointer of type int to point to the array **arr**.

int \*p;

p = arr;

or p = &arr[0]; //both the statements are equivalent.

Now we can access every element of array **arr** using **p++** to move from one element to another.

**NOTE :** You cannot decrement a pointer once incremented. p-- won't work.

#### Pointer to Array

As studied above, we can use a pointer to point to an Array, and then we can use that pointer to access the array. Lets have an example,

int i;

int a[5] = {1, 2, 3, 4, 5};

int \*p = a; *// same as int\*p = &a[0]*

for (i=0; i<5; i++)

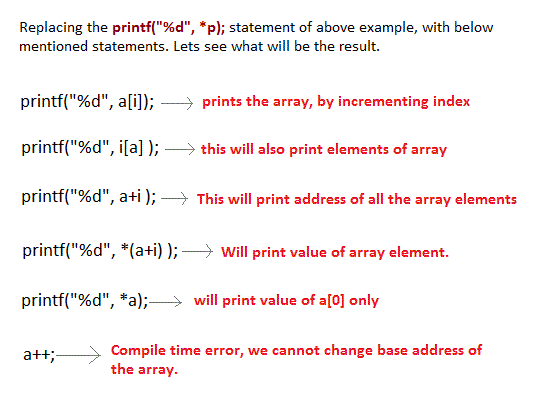
{

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

p++;

}

In the above program, the pointer **\*p** will print all the values stored in the array one by one. We can also use the Base address (**a** in above case) to act as pointer and print all the values.



#### Pointer to Multidimensional Array

A multidimensional array is of form, a[i][j]. Lets see how we can make a pointer point to such an array. As we know now, name of the array gives its base address. In a[i][j], **a** will give the base address of this array, even a+0+0 will also give the base address, that is the address of **a[0][0]** element.

Here is the generalized form for using pointer with multidimensional arrays.

**\*(\*(ptr + i) + j)**

*is same as*

**a[i][j]**

#### Pointer and Character strings

Pointer can also be used to create strings. Pointer variables of **char** type are treated as string.

char \*str = "Hello";

This creates a string and stores its address in the pointer variable **str**. The pointer **str** now points to the first character of the string "Hello". Another important thing to note that string created using **char** pointer can be assigned a value at **runtime**.

char \*str;

str = "hello"; *//thi is Legal*

The content of the string can be printed using printf() and puts().

printf("%s", str);

puts(str);

Notice that **str** is pointer to the string, it is also name of the string. Therefore we do not need to use indirection operator \*.

#### Array of Pointers

We can also have array of pointers. Pointers are very helpful in handling character array with rows of varying length.

char \*name[3]={

"Adam",

"chris",

"Deniel"

};

*//Now see same array without using pointer*

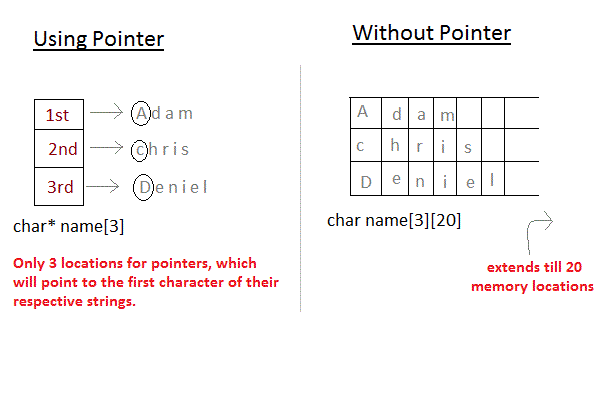
char name[3][20]= {

"Adam",

"chris",

"Deniel"

};



In the second approach memory wastage is more, hence it is prefered to use pointer in such cases.

### Pointer Arithmetic

Pointer arithmetic is very important to understand, if you want to have complete knowledge of pointer. In this topic we will study how the memory addresses change when you increment a pointer.

#### 16 bit Machine ( Turbo C )

In a 16 bit machine, size of all types of pointer, be it int\*, float\*, char\* or double\* is always **2 bytes**. But when we perform any arithmetic function like increment on a pointer, changes occur as per the size of their primitive data type.

**Size of datatypes on 16-bit Machine :**

|  |  |
| --- | --- |
| **Type** | **Size(bytes)** |
| int or signed int | 2 |
| char | 1 |
| long | 4 |
| float | 4 |
| double | 8 |
| long double | 10 |

#### Examples for Pointer Arithmetic

Now lets take a few examples and understand this more clearly.

int\* i;

i++;

In the above case, pointer will be of 2 bytes. And when we increment it, it will increment by 2 bytes because **int** is also of 2 bytes.

float\* i;

i++;

In this case, size of pointer is still 2 bytes. But now, when we increment it, it will increment by 4 bytes because **float** is of 4 bytes.

double\* i;

i++;

Similarly, in this case, size of pointer is still 2 bytes. But now, when we increment it, it will increment by 8 bytes because its data type is **double**.

#### 32 bit Machine ( Visual Basic C++ )

The concept of pointer arithmetic remains exact same, but the size of pointer and various datatypes is different in a 32 bit machine. **Pointer in 32 bit machine is of 4 bytes.**

And, following is a table for **Size of datatypes on 32-bit Machine :**

|  |  |
| --- | --- |
| **Type** | **Size(bytes)** |
| int or signed int | 4 |
| char | 2 |
| long | 8 |
| float | 8 |
| double | 16 |

**Pointer as Function parameter**

Pointer in function parameter list is used to hold address of argument passed during function call. This is also known as **call by reference**. When a function is called by reference any change made to the reference variable will effect the original variable.

**Example: Sorting an array using Pointer**

#include <stdio.h>

#include <conio.h>

void sorting(int \*x, int y);

void main()

{

int a[5],b,c;

clrscr();

printf("enter 5 nos");

for(b=0; b<5; b++)

{

scanf("%d",&a[b]);

}

sorting(a, 5);

getch();

}

void sorting(int \*x, int y)

{

inti,j,temp;

for(i=1; i<=y-1; i++)

{

for(j=0; j\*(x+j+1))

{

temp=\*(x+j);

\*(x+j)=\*(x+j+1);

\*(x+j+1)=temp;

}

}

}

for(i=0; i<5; i++)

{

printf("\t%d",\*(x+i));

}

}

**Function returning Pointer**

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function, hence if you return a pointer connected to a local variable, that pointer be will pointing to nothing when function ends.

#include <stdio.h>

#include <conio.h>

int\* larger(int\*, int\*);

void main()

{

int a=15;

int b=92;

int \*p;

p=larger(&a, &b);

printf("%d is larger",\*p);

}

int\* larger(int \*x, int \*y)

{

if(\*x > \*y)

return x;

else

return y;

}

**Safe ways to return a valid Pointer.**

1. Either use **argument with functions**. Because argument passed to the functions are declared inside the calling function, hence they will live outside the function called.
2. Or, use **static local variables** inside the function and return it. As static variables have a lifetime until main() exits, they will be available througout the program.

**Pointer to functions**

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

*type* (**\*pointer-name**)(*parameter*);

**Example :**

int (\*sum)(); //legal declaraction of pointer to function

int \*sum(); //This is not a declaraction of pointer to function

A function pointer can point to a specific function when it is assigned the name of the function.

int sum(int, int);

int (\*s)(int, int);

s = sum;

**s** is a pointer to a function **sum**. Now **sum** can be called using function pointer **s** with the list of parameter.

s (10, 20);

**Example of Pointer to Function**

#include <stdio.h>

#include <conio.h>

int sum(int x, int y)

{

returnx+y;

}

int main( )

{

int (\*fp)(int, int);

fp = sum;

int s = fp(10, 15);

printf("Sum is %d",s);

getch();

return 0;

}

Output : 25