Pointers

Introduction

dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect

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.

Consider the following example, which prints the address of the variables defined −

```
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

What are Pointers?

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.

The general form of a pointer variable declaration is −

type *var-name;

Here, **type** is the pointer's base type;

it must be a valid C data type and **var-name** is the name of the pointer variable.

The asterisk * used to declare a pointer is the same asterisk used for multiplication.

Take a look at some of the valid pointer declarations −

int *ip;

double *dp;

How to Use Pointers?

This is done by using unary operator ***** that returns the value of the variable located at the address specified by its operand.

The following example makes use of these operations −

```
#include <stdio.h>

void main () {

   int  var = 20;
   int  *ip;

   ip = &var;

   printf("Address of var variable: %x\n", &var  );


   printf("Address stored in ip variable: %x\n", ip );


   printf("Value of *ip variable: %d\n", *ip );

   getch();
}
```

Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20

NULL Pointers

A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries.

Consider the following program −

Live Demo

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;
```

```
 printf("The value of ptr is : %x\n", ptr  );

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

The value of ptr is 0

To check for a null pointer, you can use an 'if' statement as follows −

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

pointer and array

A simple example to print the address of array elements

```
#include <stdio.h>
int main( )
{
  int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;

  for ( int i = 0 ; i < 7 ; i++ )
  {

    printf("val[%d]: value is %d and address is %d\n", i, val[i], &val[i]);
  }
  return 0;
}
```

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

  int *p;


  int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;


  p = &val[0];

  for ( int i = 0 ; i<7 ; i++ )
  {
    printf("val[%d]: value is %d and address is %p\n", i, *p, p);


    p++;
```

```
  }
  Getch();
}
```

Pointers and function

Pointer as a function parameter is used to hold addresses of arguments 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 Time: Swapping two numbers using Pointer

```
#include <stdio.h>

void swap(int *a, int *b);

int main()
{
   int m = 10, n = 20;
   printf("m = %d\n", m);
   printf("n = %d\n\n", n);

   swap(&m, &n);    //passing address of m and n to the swap function
   printf("After Swapping:\n\n");
   printf("m = %d\n", m);
   printf("n = %d", n);
   return 0;
}

/*
   pointer 'a' and 'b' holds and
   points to the address of 'm' and 'n'
*/
void swap(int *a, int *b)
{
```

```
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

```
m = 10
n = 20
After Swapping:
m = 20
n = 10
```

```
#include<stdio.h>

Void exch(int *x,int *y);

Void main()

{

Int a,b;

Printf("enter two integer nos=");

Scanf("%d %d",&a,&b);

Exch(&a,&b);

Printf("%d  %d",a,b);

Getch(0;

}

Void exch(int *x,int *y)

{

        Z=*y;

        *y=*x;
```

```
        *x=z;

}
```

```
a = 10
b = 20
After Swapping:
a = 20
b = 10
```

Functions returning Pointer variables

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. They have scope only inside the function.

Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

```
#include <stdio.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;
```

```
}
```

92 is larger

Pointer and string

```
#include <stdio.h>

int main(void)

{

char str[6] = "Hello";

char *ptr = str;

 while(*ptr != '\0')

{

                printf("%c", *ptr);

               ptr++;

}

   return 0;

}

#include <stdio.h>

int main(void

 {

                char *strPtr = "Hello";

                char *t = strPtr;

                while(*t != '\0') {

               printf("%c", *t);
```

```
        t++;
 }
return 0;
}
```

Pointer and Structure

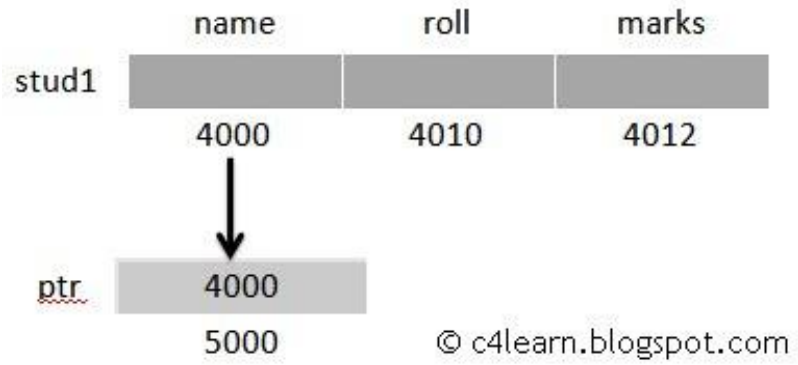
1. Address of Pointer variable can be obtained using **'&'** operator.
2. **Address** of such Structure can be assigned to the **Pointer variable** .
3. **Pointer Variable** which stores the **address of Structure** must be declared as **Pointer to Structure** .

```
#include<stdio.h>
Struct stud
{
        Char name[10];
        Int rollno;
};
Void main()
{
        Struct stud s1;
        Struct stud  *p;
        P=&s1;
        Printf("enter student name=");
        Scanf("%s",&p->name);
        Printf("enter roll no=");
        Scanf("%d",&p->rollno);
        Printf("student name=%s",p->name);
```

```
        Printf("roll no=%d",p->rollno);
        Getch();
}
```

Pointer to Structure Syntax :

```
struct student_database
{
   char name[10];
   int roll;
   int marks;
}stud1;
struct student_database *ptr;
ptr = &stud1;
```



How to Access Structure Members :

Way 1 : Using Structure Name

```
Accessing Roll Number : stud1.roll
Accessing Name        : stud1.name
```

We can use DOT operator to access the members of the structure.

Live Example 1 :

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

```
{
struct student_database
{
   char name[10];
   int roll;
   int marks;
}stud1;

stud1.roll = 10;
stud1.marks = 90;

printf("Roll Number : %d",stud1.roll);
printf("Marks of Student : %d",stud1.marks);

return 0;
}
```

Output :

```
Roll Number : 10
Marks of Student : 90
```

Way 2 : Using Indirection Operator and Pointer

```
Accessing Roll Number : (*ptr).roll
Accessing Name        : (*ptr).name
```

Pointer variable is declared and pointer to structure.

Whenever we declare pointer variable ,address of the structure variable is assigned to the pointer variable.

Live Example 2 :

```
#include <stdio.h>
int main(int argc, char *argv[])
{

struct student
```

```
 {
    char name[10];
    int roll;
    int marks;
}stud1 = {"Pritesh",90,90};

struct student *ptr;
ptr = &stud1;

printf("Roll Number : %d",(*ptr).roll);
printf("Marks of Student : %d",(*ptr).marks);

return 0;
}
```

Output of the Program :

```
Roll Number : 90
Marks of Student : 90
```

Way 3 : Using Membership Operator

```
Accessing Roll Number : ptr->roll;
Accessing Name        : ptr->name;
```

Live Example 3 :

```
#include <stdio.h>
int main(int argc, char *argv[])
{

struct student {
    char name[10];
    int roll;
    int marks;
}stud1 = {"Pritesh",90,90};
```

```
struct student *ptr;
ptr = &stud1;

printf("Roll Number : %d",(ptr)->roll);
printf("Marks of Student : %d",(ptr)->marks);

return 0;
}
```

Whenever we access the member of the structure using the pointer we use arrow operator to access the member of structure.

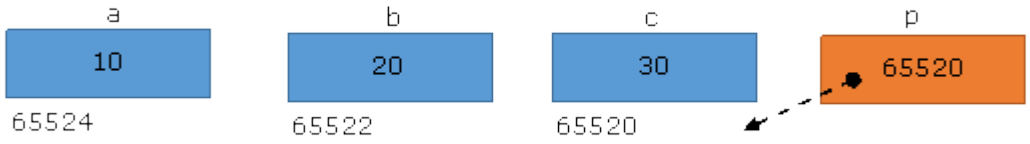
Pointer arithmetic – Scale factor

Pointer arithmetic

In case of primitive types adding 1 to the variable results incrementing of its value by 1 and subtracting 1 to the variable results decrementing of its value by 1.

Since pointer is not a normal variable adding 1 to the pointer results next variable's address. Subtracting 1 to the pointer results previous variables address.

Now we will observe the same with an example. Say for example a series of three variables of short type are declared and a pointer is stored with the address of a variable




```
1  short a=10,b=20,c=30;
2  short *p=&c;
```

Here p is the pointer to the variable c, holds the address of c 65520. Now p+1 gives the address of next variable b 65522 and p+2 gives the address of next variable a 65524.

```
#include<stdio.h>

int main()

{

short a=10,b=20,c=30;

short *p=&c;

printf("\np     %u",p);

printf("\np+1 %u",p+1);

printf("\np+2 %u",p+2);

return 0;

}
```