- ## Operations on Pointers

Pointers are variables that contain the memory address of another variable. Since an address in a memory is a numeric value we can perform arithmetic operations on the pointer values.

The different operations that can be possibly performed on **pointers** are:

Incrementing/Decrementing a pointer

Addition/Subtraction of a constant number to a pointer

Subtraction of one pointer from another

Comparison of two pointers

Addition Operation on Pointer

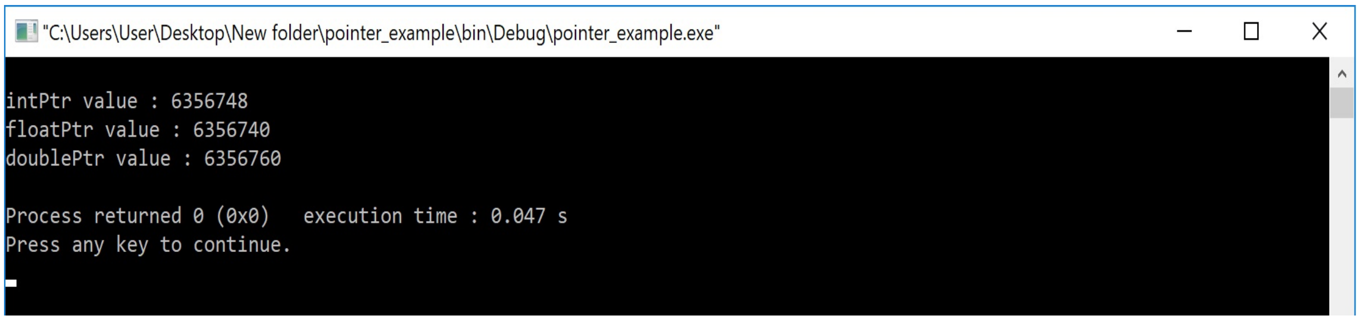
In the c programming language, the addition operation on pointer variables is calculated using the following formula...

AddressAtPointer + (NumberToBeAdd * BytesOfMemoryRequiredByDatatype)

Example Program

```
void main()
{
  int a, *intPtr ;
  float b, *floatPtr ;
  double c, *doublePtr ;
  clrscr() ;
  intPtr = &a ;  // Asume address of a is 1000
  floatPtr = &b ;  // Asume address of b is 2000
  doublePtr = &c ;  // Asume address of c is 3000

  intPtr = intPtr + 3 ;  // intPtr = 1000 + ( 3 * 2 )
  floatPtr = floatPtr + 2 ;  // floatPtr = 2000 + ( 2 * 4 )
  doublePtr = doublePtr + 5 ;  // doublePtr = 3000 + ( 5 * 6 )

  printf("intPtr value : %u\n", intPtr) ;
  printf("floatPtr value : %u\n", floatPtr) ;
  printf("doublePtr value : %u", doublePtr) ;

  getch() ;
}
```

Output:


"C:\Users\User\Desktop\New folder\pointer_example\bin\Debug\pointer_example.exe"

```
intPtr value : 6356748
floatPtr value : 6356740
doublePtr value : 6356760

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

- **Subtraction Operation on Pointer**

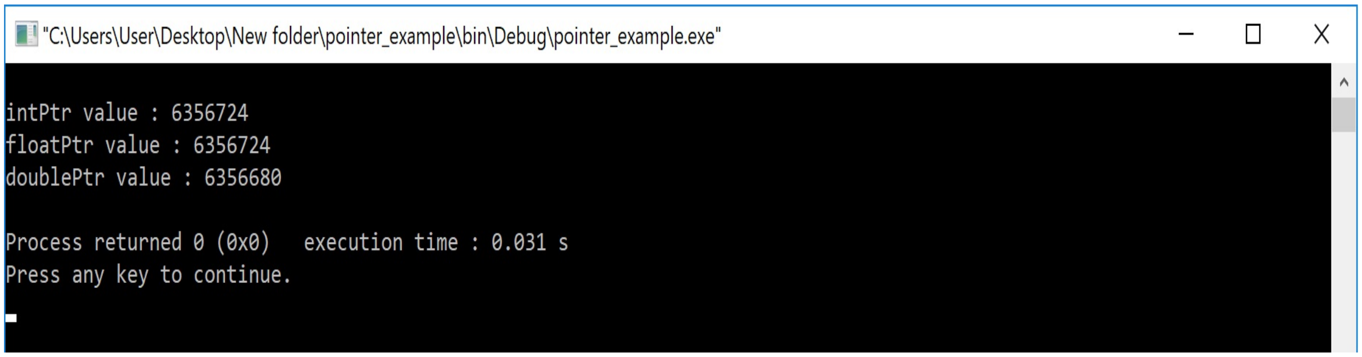
In the c programming language, the subtraction operation on pointer variables is calculated using the following formula...

AddressAtPointer - (NumberToBeAdd * BytesOfMemoryRequiredByDatatype)

Example Program

```
void main()
{
   int a, *intPtr ;
   float b, *floatPtr ;
   double c, *doublePtr ;
   clrscr() ;
   intPtr = &a ;  // Asume address of a is 1000
   floatPtr = &b ;  // Asume address of b is 2000
   doublePtr = &c ;  // Asume address of c is 3000

   intPtr = intPtr - 3 ;  // intPtr = 1000 - ( 3 * 2 )
   floatPtr = floatPtr - 2 ;  // floatPtr = 2000 - ( 2 * 4 )
   doublePtr = doublePtr - 5 ;  // doublePtr = 3000 - ( 5 * 6 )

   printf("intPtr value : %u\n", intPtr) ;
   printf("floatPtr value : %u\n", floatPtr) ;
   printf("doublePtr value : %u", doublePtr) ;

   getch() ;
}
```

Output:


"C:\Users\User\Desktop\New folder\pointer_example\bin\Debug\pointer_example.exe"

```
intPtr value : 6356724
floatPtr value : 6356724
doublePtr value : 6356680

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Increment & Decrement Operation on Pointer

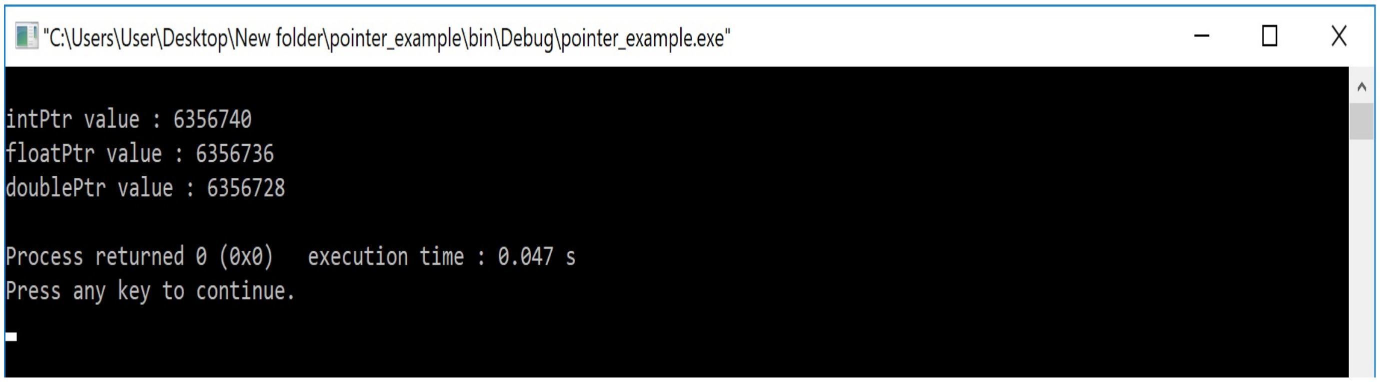
The increment operation on pointer variable is calculated as follows...

AddressAtPointer + NumberOfBytesRequiresByDatatype

Example Program

```
void main()
{
   int a, *intPtr ;
   float b, *floatPtr ;
   double c, *doublePtr ;
   clrscr() ;
   intPtr = &a ;  // Asume address of a is 1000
   floatPtr = &b ;  // Asume address of b is 2000
   doublePtr = &c ;  // Asume address of c is 3000

   intPtr++ ;  // intPtr = 1000 + 2
   floatPtr++ ;  // floatPtr = 2000 + 4
   doublePtr++ ;  // doublePtr = 3000 + 6

   printf("intPtr value : %u\n", intPtr) ;
   printf("floatPtr value : %u\n", floatPtr) ;
   printf("doublePtr value : %u", doublePtr) ;

   getch() ;
}
```

Output:


"C:\Users\User\Desktop\New folder\pointer_example\bin\Debug\pointer_example.exe"

```
intPtr value : 6356740
floatPtr value : 6356736
doublePtr value : 6356728

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

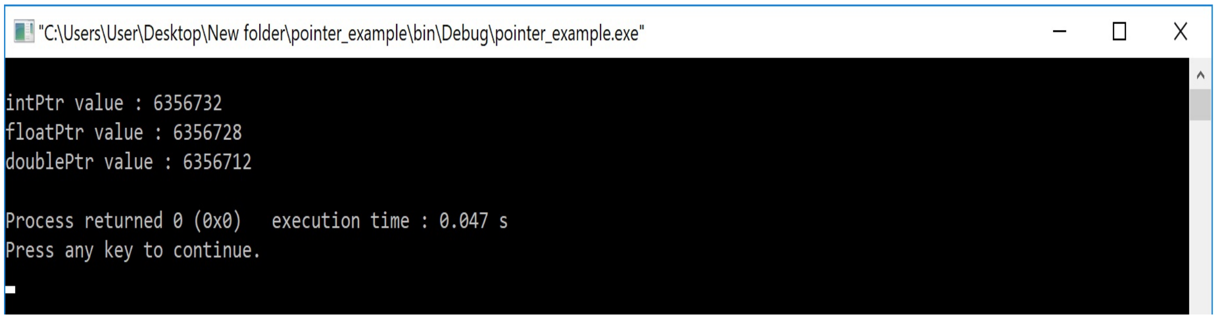
The decrement operation on pointer variable is calculated as follows...

AddressAtPointer - NumberOfBytesRequiresByDatatype

Example Program

```
void main()
{
   int a, *intPtr ;
   float b, *floatPtr ;
   double c, *doublePtr ;
   clrscr() ;
   intPtr = &a ;  // Asume address of a is 1000
   floatPtr = &b ;  // Asume address of b is 2000
   doublePtr = &c ;  // Asume address of c is 3000

   intPtr-- ;  // intPtr = 1000 - 2
   floatPtr-- ;  // floatPtr = 2000 - 4
   doublePtr-- ;  // doublePtr = 3000 - 6

   printf("intPtr value : %u\n", intPtr) ;
   printf("floatPtr value : %u\n", floatPtr) ;
   printf("doublePtr value : %u", doublePtr) ;

   getch() ;
}
```

Output:



"C:\Users\User\Desktop\New folder\pointer_example\bin\Debug\pointer_example.exe"

```
intPtr value : 6356732
floatPtr value : 6356728
doublePtr value : 6356712

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Comparison of Pointers

The comparison operation is perform between the pointers of same datatype only. In c programming language, we can use all comparison operators (relational operators) with pointers.

We can't perform multiplication and division operations on pointers.

Pointers to Pointers in C

In the c programming language, we have pointers to store the address of variables of any datatype. A pointer variable can store the address of a normal variable. C programming language also provides a pointer variable to store the address of another pointer variable. This type of pointer variable is called a pointer to pointer variable. Sometimes we also call it a double pointer. We use the following syntax for creating pointer to pointer…

datatype **pointerName ;

Example Program

```
int **ptr ;
```

Here, **ptr** is an integer pointer variable that stores the address of another integer pointer variable but does not stores the normal integer variable address.

MOST IMPORTANT POINTS TO BE REMEMBERED

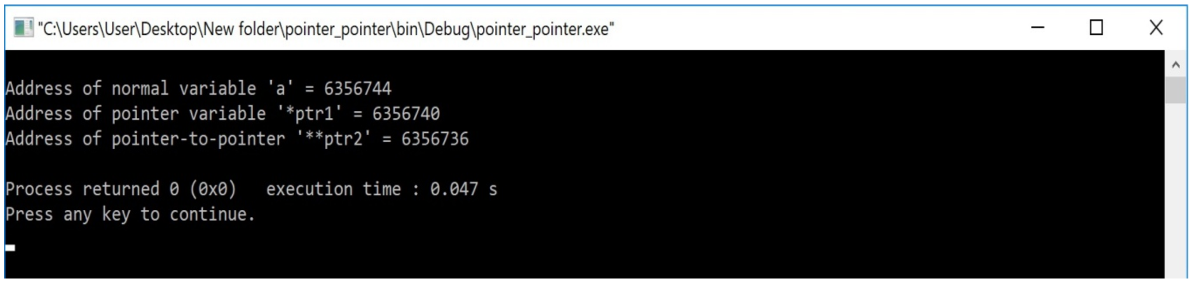
1. To store the address of normal variable we use single pointer variable
2. To store the address of single pointer variable we use double pointer variable
3. To store the address of double pointer variable we use triple pointer variable
4. Similarly the same for remaining pointer variables also…

Example Program

```
#include<stdio.h>
```

```
#include<conio.h>

int main()
{
   int a ;
   int *ptr1 ;
   int **ptr2 ;
   int ***ptr3 ;

   ptr1 = &a ;
   ptr2 = &ptr1 ;
   ptr3 = &ptr2 ;

   printf("\nAddress of normal variable 'a' = %u\n", ptr1) ;
   printf("Address of pointer variable '*ptr1' = %u\n", ptr2) ;
   printf("Address of pointer-to-pointer '**ptr2' = %u\n", ptr3) ;
    return 0;
}
```

Output:


"C:\Users\User\Desktop\New folder\pointer_pointer\bin\Debug\pointer_pointer.exe"

```
Address of normal variable 'a' = 6356744
Address of pointer variable '*ptr1' = 6356740
Address of pointer-to-pointer '**ptr2' = 6356736

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Pointers to void in C

In the c programming language, pointer to void is the concept of defining a pointer variable that is independent of data type. In C programming language, a void pointer is a pointer variable used to store the address of a variable of any datatype. That means single void pointer can be used to store the address of integer variable, float variable, character variable, double variable or any structure variable. We use the keyword **"void"** to create void pointer. We use the following syntax for creating a pointer to void…

void *pointerName ;

Example Code

```
void *ptr ;
```

Here, **"ptr"** is a void pointer variable which is used to store the address of any datatype variable.

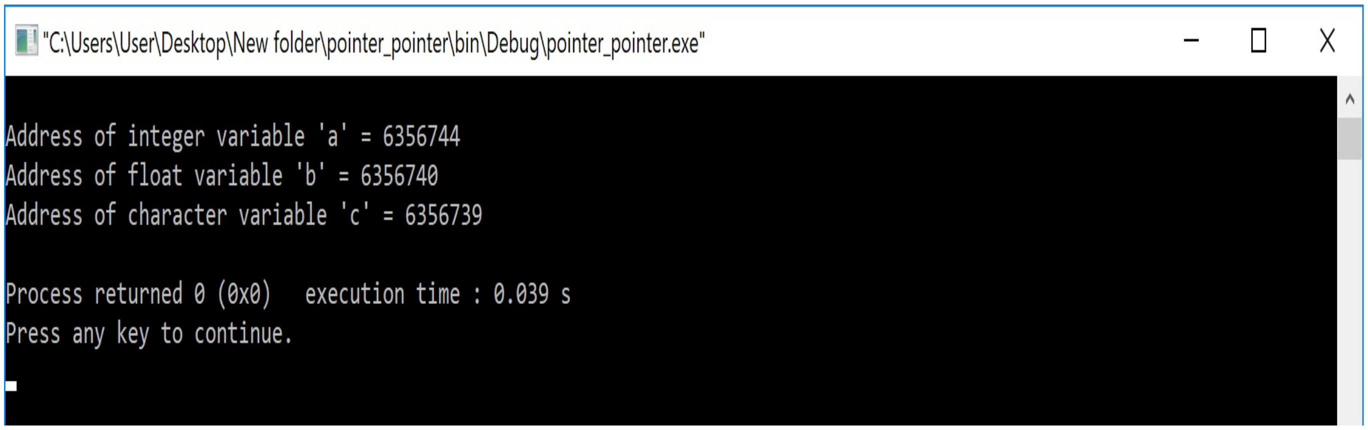
MOST IMPORTANT POINTS TO BE REMEMBERED

1. void pointer stores the address of any datatype variable.

Example Program

```
#include<stdio.h>
#include<conio.h>

int main()
{
   int a ;
   float b ;
   char c ;

   void *ptr ;

   clrscr() ;

   ptr = &a ;
   printf("Address of integer variable 'a' = %u\n", ptr) ;

   ptr = &b ;
   printf("Address of float variable 'b' = %u\n", ptr) ;

   ptr = &c ;
   printf("Address of character variable 'c' = %u\n", ptr) ;
   return 0;
}
```

Output:



"C:\Users\User\Desktop\New folder\pointer_pointer\bin\Debug\pointer_pointer.exe"

```
Address of integer variable 'a' = 6356744
Address of float variable 'b' = 6356740
Address of character variable 'c' = 6356739

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```