Pointers and Arrays

```
# include <stdio.h>
int main( )
{ int i = 3, *x ;
float j = 1.5, *y ;
char k = 'c', *z ;
printf ( "Value of i = %d\n", i ) ;
printf ( "Value of j = %f\n", j ) ;
printf ( "Value of k = %c\n", k ) ;
x = &i ;
y = &j ;
z = &k ;
printf ( "Original address in x = %u\n", x ) ;
printf ( "Original address in y = %u\n", y ) ;
printf ( "Original address in z = %u\n", z ) ;
x++ ;
y++ ;
z++ ;
printf ( "New address in x = %u\n", x ) ;
printf ( "New address in y = %u\n", y ) ;
printf ( "New address in z = %u\n", z ) ;
return 0 ; }
```

Pointers and Arrays

```
# include <stdio.h>
int main( )
{ int i = 3, *x ;
float j = 1.5, *y ;
char k = 'c', *z ;
printf ( "Value of i = %d\n", i ) ;
printf ( "Value of j = %f\n", j ) ;
printf ( "Value of k = %c\n", k ) ;
x = &i ;
y = &j ;
z = &k ;
printf ( "Original address in x = %u\n", x ) ;
printf ( "Original address in y = %u\n", y ) ;
printf ( "Original address in z = %u\n", z ) ;
x++ ;
y++ ;
z++ ;
printf ( "New address in x = %u\n", x ) ;
printf ( "New address in y = %u\n", y ) ;
printf ( "New address in z = %u\n", z ) ;
return 0 ; }
```

Output:

```
Value of i = 3
Value of j = 1.500000
Value of k = c
Original address in x = 65524
Original address in y = 65520
Original address in z = 65519
New address in x = 65528
New address in y = 65524
New address in z = 65520
```

Operations on Pointers

- (a) Addition of a number to a pointer. For example,

```
int i = 4, *j, *k ;
j = &i ;
j = j + 1 ;
j = j + 9 ;
k = j + 3 ;
```

- (b) Subtraction of a number from a pointer. For example,

```
int i = 4, *j, *k ;
j = &i ;
j = j - 2 ;
j = j - 5 ;
k = j - 6 ;
```

(c) Subtraction of one pointer from another.

- One pointer variable can be subtracted from another provided both variables point to elements of the same array. The resulting value indicates the number of elements separating the corresponding array elements.

```
# include <stdio.h>
int main( )
{
int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
int *i, *j ;
i = &arr[ 1 ] ;
j = &arr[ 5 ] ;
printf ( "%d %d\n", j - i, *j - *i ) ;
return 0 ;
}
```

(d) Comparison of two pointer variables

- Pointer variables can be compared provided both variables point to objects of the same data type.
- Such comparisons can be useful when both pointer variables point to elements of the same array.
- The comparison can test for either equality or inequality.

```
# include <stdio.h>
int main( )
{
int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
int *j, *k ;
j = &arr [ 4 ] ;
k = ( arr + 4 ) ;
if ( j == k )
printf ( "The two pointers point to the same location\n" ) ;
else
printf ( "The two pointers do not point to the same location\n" ) ;
return 0 ;
}
```

&arr[0] =arr

A word of caution! Do not attempt the following operations on pointers... they would never work out.

(a) Addition of two pointers

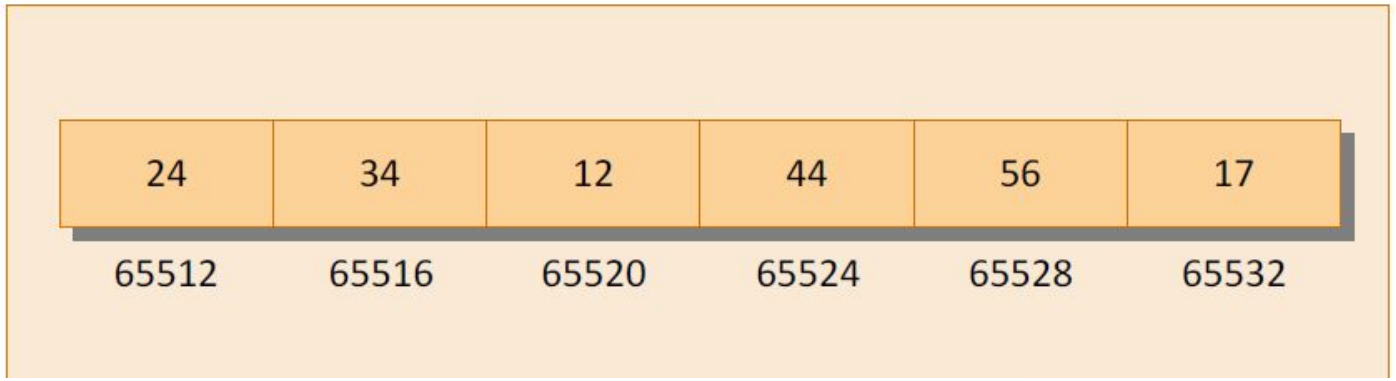
(b) Multiplication of a pointer with a constant

(c) Division of a pointer with a constant

- Now we will try to correlate the following two facts, which we have learnt :

(a) Array elements are always stored in contiguous memory locations.

(b) A pointer when incremented always points to an immediately next location of its type.




```
int  num[ ] = { 24, 34, 12, 44, 56, 17 } ;
```

1. On mentioning the name of the array, we get its base address. Thus, by saying ***num**, we would be able to refer to the zeroth element of the array, that is, 24.
2. One can easily see that ***num** and ***(num + 0)** both refer to 24.
3. Similarly, by saying ***(num + 1)**, we can refer the first element of the array, that is, 34. In fact, this is what the C compiler does internally.
4. When we say, **num[i]**, the C compiler internally converts it to ***(num + i)**. This means that all the following notations are same:

num[i]
*(num + i)
*(i + num)
i[num]

Arrays as Function arguments:

- **Passing array to function:**
- Array can be passed to function by two ways:
- 1. **Pass Entire array**
- 2. **Pass Array element by element**

1. Pass Entire array

- Here entire array can be passed as a argument to function.
- Function gets **complete access** to the original array.
- While passing entire array; address of first element is passed to function, any changes made inside function, directly **affects the Original value**.
- Function Passing method : **“Pass by Address“**

2. Pass Array element by element

- Here individual elements are passed to function as argument.
- Duplicate **carbon copy of Original variable** is passed to function.
- So any changes made inside function **do not affect the original value.**
- Function doesn’t get complete access to the original array element.
- Function passing method is **“Pass by Value“**

Passing entire array to function:

- Parameter Passing Scheme : **Pass by Reference**
- Pass name of array as function parameter.
- Name contains the base address i.e. (Address of 0th element)
- Array values are updated in function.
- Values are reflected inside main function also

Passing Array to a function

- Passing element by element (Call by Value)
- Passing element by element (Call by Reference)
- Passing Complete Array

Example

- Write a program to copy the contents of one array into another in the reverse order.
  - Using Call by Value – Passing element by element,
  - Call by Reference – Passing element by element,
  - Passing Complete Array

Call by Value (Element by Element)

Passing Array Elements to a Function using Call by Value

```
/* Demonstration of call by value */
main( )
{
   int i ;
   int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
   for ( i = 0 ; i <= 6 ; i++ )
         display ( marks[i] ) ;
 }
display ( int m )
{
   printf ( "%d ", m ) ;
}
```

// Function Call

// Function Definition

Passing Array Elements to a Function using Call by Value

- Here, we are passing an individual array element at a time to the function **display( )** and getting it printed in the function **display( ).**
- Note that since at a time only one element is being passed, this element is collected in an ordinary integer variable **m**, in the function **display( )**.

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
          printf("\nEnter the value of a[%d] ",i);
          scanf("%d",&a[i]);
    }
return 0;
}
```

Write a program to copy the contents of one array into another in the reverse order.

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
         reverse(a[i],i);
    return 0;
}
```

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
        printf("\nEnter the value of a[%d] ",i);
        scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
        reverse(a[i],i);
    return 0;
}
```

```
void reverse(int j, int x)
{
      int rev[5]; //array initialized size of the
array
      printf("rev[%d] =%d\n", 4-x,j);
}
```

A[0]	A[1]	A[2]	A[3]	A[4]
10	20	30	40	50

A[i] , i	4-x, j	Rev[4-x] , j
50, 4	0, 50	Rev[0] =50
40,3	1,40	Rev[1]=40
30,2	2, 30	Rev[2] =30
20, 1	1, 20	Rev[1]=20
10,0	0, 10	Rev[0]=10

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
         reverse(a[i],i);
    return 0;
}
```

```
void reverse(int j, int x)
{
      int rev[5]; //array initialized size of the
array
      printf("rev[%d] =%d\n", 4-x,j);
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	7	5	3
1234	1236	1238	1240	1242

j
3
4598

i	reverse(a[i],i) reverse(value, index)	Rev[4-index, value)
4	reverse(a[4],4) reverse(3,4)	rev[4-4] = 3 i.e rev[0]=3

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
         reverse(a[i],i);
    return 0;
}
```

```
void reverse(int j, int x)
{
      int rev[5]; //array initialized size of the
array
      printf("rev[%d] =%d\n", 4-x,j);
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	7	5	3
1234	1236	1238	1240	1242

j
5
4600

i	reverse(a[i],i) reverse(value, index)	Rev[4-index, value)
4	reverse(a[4],4) reverse(3,4)	rev[4-4] = 3 i.e rev[0]=3
3	reverse(a[i],i) reverse(5,3)	rev[4-3] = 5 i.e rev[1]=5

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
         reverse(a[i],i);
    return 0;
}
```

```
void reverse(int j, int x)
{
      int rev[5]; //array initialized size of the
array
      printf("rev[%d] =%d\n", 4-x,j);
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	7	5	3
1234	1236	1238	1240	1242

j
7
4602

i	reverse(a[i],i) reverse(value, index)	Rev[4-index, value)
4	reverse(a[4],4) reverse(3,4)	rev[4-4] = 3 i.e rev[0]=3
3	reverse(a[i],i) reverse(5,3)	rev[4-3] = 5 i.e rev[1]=5
2	reverse(a[i],i) (reverse(7,2)	rev[4-2] = 7 i.e rev[2]=7

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
         reverse(a[i],i);
    return 0;
}
```

```
void reverse(int j, int x)
{
      int rev[5]; //array initialized size of the
array
      printf("rev[%d] =%d\n", 4-x,j);
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	7	5	3
1234	1236	1238	1240	1242

j
4
4604

i	reverse(a[i],i) reverse(value, index)	Rev[4-index, value)
4	reverse(a[4],4) reverse(3,4)	rev[4-4] = 3 i.e rev[0]=3
3	reverse(a[i],i) reverse(5,3)	rev[4-3] = 5 i.e rev[1]=5
2	reverse(a[i],i) (reverse(7,2)	rev[4-2] = 7 i.e rev[2]=7
1	reverse(a[i],i) (reverse 4, 1)	rev[4-1] = 4 i.e rev[3]=4

Call by Value (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i >= 0;  i--)
         reverse(a[i],i);
    return 0;
}
```

```
void reverse(int j, int x)
{
      int rev[5]; //array initialized size of the
array
      printf("rev[%d] =%d\n", 4-x,j);
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	7	5	3
1234	1236	1238	1240	1242

j
2
4606

i	reverse(a[i],i) reverse(value, index)	Rev[4-index, value)
4	reverse(a[4],4) reverse(3,4)	rev[4-4] = 3 i.e rev[0]=3
3	reverse(a[i],i) reverse(5,3)	rev[4-3] = 5 i.e rev[1]=5
2	reverse(a[i],i) (reverse(7,2)	rev[4-2] = 7 i.e rev[2]=7
1	reverse(a[i],i) (reverse 4, 1)	rev[4-1] = 4 i.e rev[3]=4
0	reverse(a[i],i) reverse(2,0)	rev[4-0] = 2 i.e rev[4]=2

Call by Reference (Element by Element)

Passing Array Elements to a Function –Using Call by Reference

```
/* Demonstration of call by reference */
main( )
{
  int i ;
  int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
  for ( i = 0 ; i <= 6 ; i++ )
      disp ( &marks[i] ) ;
}
disp ( int *n )
{
   printf ( "%d ", *n ) ;
 }
```

Passing Array Elements to a Function –Using Call by Reference

- Here, we are passing addresses of individual array elements to the function **display( )**.
- Hence, the variable in which this address is collected (**n**) is declared as a pointer variable.
- And since **n** contains the address of array element, to print out the array element we are using the ‘value at address’ operator **(*)**.

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }

    return 0;
}
```

Write a program to reverse the contents of the array.
(Reversed array appears in the same array)

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i > 2;  i--)
         reverse(&a[0],i);
    return 0;
}
```

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i > 2;  i--)
         reverse(&a[0],i);
for(i=0;i<5;i++)
            printf("%d",a[i]);
    return 0;
}
```

```
void reverse(int  *j, int x)
{
   int temp,y;
   y = 4-x;
   temp =*(j+x);
   *(j+x) = *(j+y);
   *(j+y) = temp;
}
```

&a[0]= 6700, 4
1,2,3,4,5
Y=0
Temp= *(6700+4) = 4

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
         printf("\nEnter the value of a[%d] ",i);
         scanf("%d",&a[i]);
    }
    for(i=4;  i > 2;  i--)
         reverse(&a[0],i);
for(i=0;i<5;i++)
            printf("%d",a[i]);
    return 0;
}
```

```
void reverse(int  *j, int x)
{
   int temp,y;
   y = 4-x;
   temp =*(j+x);
   *(j+x) = *(j+y);
   *(j+y) = temp;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	7	5	3
2316	2318	2320	2322	2324

i	Base Address = 2316
4	X = 4, y = 0 ,
	(J+ x) = 2324; *(j+x) =3
	(j+y) = 2316; *(j+y) =2

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
          printf("\nEnter the value of a[%d] ",i);
          scanf("%d",&a[i]);
    }
    for(i=4;  i > 2;  i--)
          reverse(&a[0],i);
for(i=0;i<5;i++)
              printf("%d",a[i]);
    return 0;
}
```

```
void reverse(int  *j, int x)
{
   int temp,y;
   y = 4-x;
   temp =*(j+x);
   *(j+x) = *(j+y);
   *(j+y) = temp;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
3	4	7	5	2
2316	2318	2320	2322	2324

i	Base Address = 2316
4	X = 4, y = 0 ,
	(J+ x) = 2324; *(j+x) =3
	(j+y) = 2316; *(j+y) =2

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
          printf("\nEnter the value of a[%d] ",i);
          scanf("%d",&a[i]);
    }
    for(i=4;  i > 2;  i--)
          reverse(&a[0],i);
for(i=0;i<5;i++)
              printf("%d",a[i]);
    return 0;
}
```

```
void reverse(int  *j, int x)
{
   int temp,y;
   y = 4-x;
   temp =*(j+x);
   *(j+x) = *(j+y);
   *(j+y) = temp;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
3	4	7	5	2
2316	2318	2320	2324	5487

i	Base Address = 2316
3	X = 3, y = 1,
	(J+ x) = 2324; *(j+x) =5
	(j+y) = 2318; *(j+y) =4

Call by Reference (Element by Element)

```
#include<stdio.h>
void reverse(int j, int x); //function declaration
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
          printf("\nEnter the value of a[%d] ",i);
          scanf("%d",&a[i]);
    }
    for(i=4;  i > 2;  i--)
          reverse(&a[0],i);
for(i=0;i<5;i++)
              printf("%d",a[i]);
    return 0;
}
```

```
void reverse(int  *j, int x)
{
   int temp,y;
   y = 4-x;
   temp =*(j+x);
   *(j+x) = *(j+y);
   *(j+y) = temp;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
3	5	7	4	2
2316	3476	4586	7621	2324

i	
3	X = 3, y = 1,
	(J+ x) = 2322; *(j+x) =5
	(j+y) = 2318; *(j+y) =4

Passing Complete Array to a function

Possible when base address is known

Accessing array elements when base address is known

```
main( )
 {
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
   printf ( "\naddress = %u ", j ) ;
   printf ( "element = %d", *j ) ;
   j++ ; /* increment pointer to point to next location */
  }
 }
```

The output of this program would be:

address = 65512 element = 24
address = 65514 element = 34
address = 65516 element = 12
address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17

Num[0]	Num[1]	Num[2]	Num[3]	Num[4]	Num[5]
24	34	12	44	56	17
65512	65514	65516	65518	65520	65522

Accessing array elements when base address is known

```
main( )
 {
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
   printf ( "\naddress = %u ", j ) ;
   printf ( "element = %d", *j ) ;
  j++ ; /* increment pointer to point to next location */
  }
 }
```

The output of this program would be:

address = 65512 element = 24
address = 65514 element = 34
address = 65516 element = 12
address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17

Num[0]	Num[1]	Num[2]	Num[3]	Num[4]	Num[5]
24	34	12	44	56	17
65512	65514	65516	65518	65520	65522

Accessing array elements when base address is known

```
main( )
 {
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
   printf ( "\naddress = %u ", j ) ;
   printf ( "element = %d", *j ) ;
  j++ ; /* increment pointer to point to next location */
  }
 }
```

The output of this program would be:

```
address = 65512 element = 24
address = 65514 element = 34
address = 65516 element = 12
address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17
```

Num[0]	Num[1]	Num[2]	Num[3]	Num[4]	Num[5]
24	34	12	44	56	17
65512	65514	65516	65518	65520	65522

Accessing array elements when base address is known

```
main( )
 {
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
   printf ( "\naddress = %u ", j ) ;
   printf ( "element = %d", *j ) ;
  j++ ; /* increment pointer to point to next location */
  }
 }
```

The output of this program would be:

```
address = 65512 element = 24
address = 65514 element = 34
address = 65516 element = 12
address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17
```

Num[0]	Num[1]	Num[2]	Num[3]	Num[4]	Num[5]
24	34	12	44	56	17
65512	65514	65516	65518	65520	65522

Accessing array elements when base address is known

```
main( )
 {
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
   printf ( "\naddress = %u ", j ) ;
   printf ( "element = %d", *j ) ;
  j++ ; /* increment pointer to point to next location */
  }
 }
```

The output of this program would be:

```
address = 65512 element = 24
address = 65514 element = 34
 address = 65516 element = 12
 address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17
```

Num[0]	Num[1]	Num[2]	Num[3]	Num[4]	Num[5]
24	34	12	44	56	17
65512	65514	65516	65518	65520	65522

Accessing array elements when base address is known

```
main( )
 {
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[0] ; /* assign address of zeroth element */
 for ( i = 0 ; i <= 5 ; i++ )
 {
   printf ( "\naddress = %u ", j ) ;
   printf ( "element = %d", *j ) ;
  j++ ; /* increment pointer to point to next location */
  }
 }
```

The output of this program would be:

```
address = 65512 element = 24
address = 65514 element = 34
 address = 65516 element = 12
 address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17
```

Num[0]	Num[1]	Num[2]	Num[3]	Num[4]	Num[5]
24	34	12	44	56	17
65512	65514	65516	65518	65520	65522

- In this program, to begin with we have collected the base address of the array (address of the 0^{th} element) in the variable **j** using the statement,
- j = &num[0] ; /* assigns address 65512 to j */
- When we are inside the loop for the first time, **j** contains the address 65512, and the value at this address is 24.
- These are printed using the statements,
  - **printf ("\naddress = %u ", j) ;**
  - **printf ("element = %d", *j) ;**

- On incrementing **j** it points to the next memory location of its type (that is location no. 65514).
- But location no. 65514 contains the second element of the array, therefore when the **printf( )** statements are executed for the second time they print out the second element of the array and its address (i.e. 34 and 65514)... and so on till the last element of the array has been printed

Passing an Entire Array to a Function

```
/* Demonstration of passing an entire array to a function */
main( )
{
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  dislpay ( &num[0], 6 ) ;
}

display ( int *j, int n )
 { int i ;
    for ( i = 0 ; i <= n - 1 ; i++ )
      {
          printf ( "\nelement = %d", *j ) ;
          j++ ; /* increment pointer to point to next element */
     } }
```

Passing an Entire Array to a Function

- Here, the **display( )** function is used to print out the array elements.
- The address of the zeroth element is being passed to the **display( )** function.
- The **for** loop is same as the one used in the earlier program to access the array elements using pointers.
- It is also necessary to pass the total number of elements in the array, otherwise the **display( )** function would not know when to terminate the **for** loop.
- Note that the address of the zeroth element (many a times called the base address) can also be passed by just passing the name of the array.
- Thus, the following two function calls are same:

```
display ( &num[0], 6 ) ;
display ( num, 6 ) ;
```

Passing Complete Array

```
#include<stdio.h>
void reverse(int *j, int x);
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
      printf("\nEnter the value of a[%d] ",i);
           scanf("%d",&a[i]);
    }
    reverse(&a[0],5);
    return 0;
}
```

```
void reverse(int *j, int x)
{
      int  i, temp,y;
      for (i=0;i<x/2;i++)
      {
            y = 5-i-1;
      temp = *(j+i);
            *(j+i) = *( j+y);
            *(j+y) =temp;

   }
}
```

Example

- Write a program which performs the following tasks:
- – initialize an integer array of 5 elements in **main( )**
- – pass the entire array to a function **modify( )**
- – in **modify( )** multiply each element of array by 3
- – return the control to **main( )** and print the new array elements in **main( )**

```
#include<stdio.h>
void modify(int *j, int x);
int main()
{
   int a[5], i;
    for(i=0;i<5;i++)
    {
     printf("\nEnter the value of a[%d] ",i);
    scanf("%d",&a[i]);
    }
    modify(a,5);
   for(i=0; i < 5;i++)
    {
    printf("\na[%d] = %d",i, a[i]);
    }
    return 0;
}
```

```
void modify(int *j, int x)
{
        int i;
        for (i=0;i<x;i++)
        {
         *j = *j *3;
          j++;
    }
}
```

Exercise

- Find the smallest number in an array
  - Using Call by Value – Passing element by element,
  - Call by Reference – Passing element by element,
  - Passing Complete Array