UNIT-IV STORAGE CLASS AND POINTERS

Contents

Storage Class Specifier - Auto, Extern, Static and Register. The Pointers – '& ' and '*' Pointers, Pointer Expressions, Arrays using pointers, Structures using pointers, Functions using pointers, Function as argument, Command Line arguments.

Storage Class Specifier

The scope of the variable specifies the part / parts in which the variable is alive. Depending on the place of the variable declared , the variables are classified into two broad categories as **Global** and **Local** variables. In some languages like BASIC, all the variables are global and the values are retained throughout the program. But in C language , the availability of value of the variable depends on the 'storage ' class of variable. In C , there are four types of storage classes. They are

1) Local or Automatic variables
2) Global or External variables
3) Static Variables
4) Register Variables.

1. Automatic variable:

An Automatic variable is a local variable which is declared inside the function. The memory cell is created at the time of declaration statement is executed and is destroyed when the flow comes out of the function. These variables are also called as the internal variables. A variable which is declared inside the function without using any storage class is assumed as the local variable because the default storage class is automatic. A variable can be declared automatic explicitly by using the keyword "auto" as

```
main ( )
{
auto int x;
…
…..
}
```

2. External variable:

External variable is a global variable which is declared outside the function. The memory cell is created at the time of declaration statement is executed and is not destroyed when the flow comes out of the function. The global variables can be accessed by any function in the same program. A global variable can be declared externally in the global variable declaration section.

```
int x = 100;
main ( )
{
…..
x = 200;
f1 ( );
f2 ( );    …
…..
}

f1 ( )
{
x = x + 1;
….
…..
}
f2 ( )
{
x = x + 100;
….
…..
}
```

The variable x is declared above all the functions. It can be accessed by all the functions as main( ) , f1( ) and f2( ). The final value of x is 301.

3. Static Variables

These variables are alive throughout the program. A variable can be declared as static by using the keyword “ **static** ”as

staic int x ;

A static variable can be initialized only once at the time of declaration. The initialization part is executed only once and retain the remainder value of the program.

Example

```
main ( )
{
void f1( );
f1 ( );
f1 ( );
f1 ( );
}
void f1( )
{
static int x = 0;
x = x + 1;
printf(“\n The Value of X is %d “,x );
}
```

```
The above program produces the following output
     The Value of X is 1
```

```
      The Value of X is 2
      The Value of X is 3.
```

4. Register Variables

If we want to store the variable in a register instead of memory , the variable can be declared as the register variables by using the keyword “register ” as

register int x;

If the variables are stored in the registers , they can be accessed faster than a memory access. So the frequently accessed variables can be declared as the register variables.

Example

```
main ( )
{
register x , y, z;
scanf(“%d%d”,&x,&y);
z=x+y;
printf(“\n The Output is %d”,z);
}
```

In the above program , all the variables are stored in the registers instead of memory.

Examples

<u>Auto</u>

eg:1

```
#include<stdio.h>
void main()
{
 auto mum = 20 ;
 {
      auto num = 60 ;
      printf("nNum : %d",num);
 }
 printf("nNum : %d",num);
}
```

Output :

```
Num : 60
Num : 20
```

Note :
Two variables are declared in different blocks, so they are treated as different variables

eg:2

```
#include<stdio.h>
void increment(void);
void main()
{
increment();
increment();
increment();
increment();
}
void increment(void)
{
auto int i = 0 ;
printf ( “%d “, i ) ;
i++;
}
```

Output:

```
0 0 0 0
```

Extern

eg:1

```
#include<stdio.h>
int num =  75 ;
void display();
void main()
{
 extern int num ;
 printf("nNum : %d",num);
 display();
}
void display()
{
 extern int num ;
 printf("nNum : %d",num);
}
```

Output :

```
Num : 75
Num : 75
```

Note :
Declaration within the function indicates that the function uses external variable
Functions belonging to same source code do not require declaration (no need to write extern). If variable is defined outside the source code, then declaration using extern keyword is required.

eg:2

```
#include<stdio.h>
int x = 10 ;
void main( )
{
extern int y;
printf("The value of x is %d \n",x);
printf("The value of y is %d",y);


}
int y=50;
Output:

The value of x is 10
The value of y is 50
Example program for register variable in C:
```

Static

eg:1

```
#include<stdio.h>
void Check();
int main(){
   Check();
   Check();
   Check();
}
void Check(){
    static int c=0;
    printf("%d\t",c);
    c+=5;
}
```

Output

```
0       5      10
```

Note:
During first function call, it will display 0. Then, during second function call, variable c will not be initialized to 0 again, as it is static variable. So, 5 is displayed in second function call and 10 in third call.
If variable c had been automatic variable, the output would have been:

```
0     0     0
```

eg:2

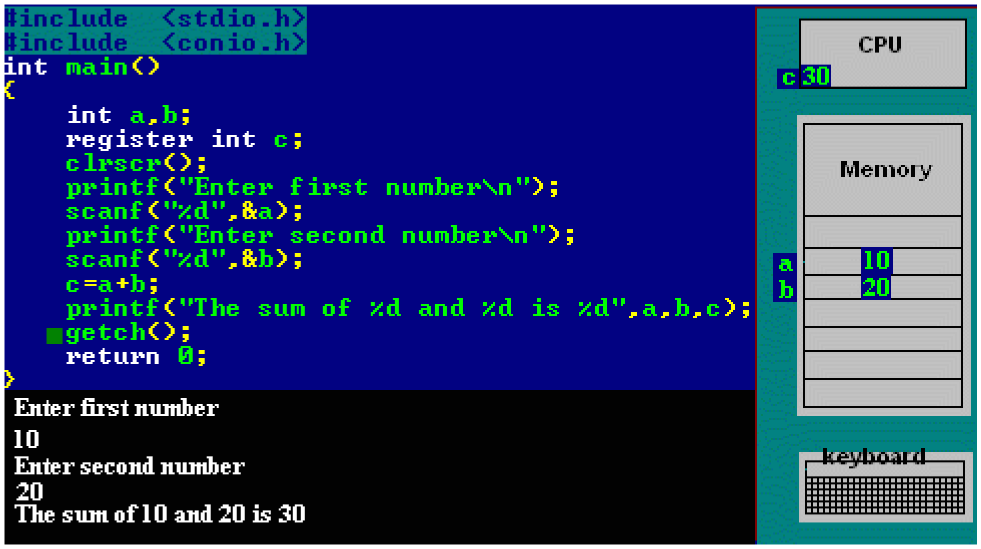
```
#include<stdio.h>
void increment(void);
int main()
{
increment();
increment();
increment();
increment();
return 0;
}
void increment(void)
{
static int i = 0 ;
printf ( “%d “, i ) ;
i++;
}
Output:

0 1 2 3
```

Register

eg:1

```
#include<stdio.h>
void main()
{
int num1,num2;
register int sum;
printf("\nEnter the Number 1 : ");
scanf("%d",&num1);
printf("\nEnter the Number 2 : ");
scanf("%d",&num2);
sum = num1 + num2;
printf("\nSum of Numbers : %d",sum);
}
```



```
#include  <stdio.h>
#include  <conio.h>
int main()
{
    int a,b;
    register int c;
    clrscr();
    printf("Enter first number\n");
    scanf("%d",&a);
    printf("Enter second number\n");
    scanf("%d",&b);
    c=a+b;
    printf("The sum of %d and %d is %d",a,b,c);
    getch();
    return 0;
}
```

```
Enter first number
10
Enter second number
20
The sum of 10 and 20 is 30
```

eg:2

```
#include <stdio.h>
int main()
{
register int i;
int arr[5];// declaring array
arr[0] = 10;// Initializing array
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
for (i=0;i<5;i++)
{
// Accessing each variable
printf("value of arr[%d] is %d \n", i, arr[i]);
}
return 0;
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

POINTERS

Definition:-

A pointer is a variable whose value is the address of another variable. Like any variables, we must declare a pointer variable at the beginning of the program. We can create pointer to any variable type as given in te below examples.

The general format of a pointer variable declaration is as follows:-

datatype *pointervariable;

```
Examples:    int *ip;              //pointer to an integer variable
             float *fp;            //pointer to a float variable
             double *dp;           //pointer to a double variable
             char *cp;             //pointer to a character variable
```

POINTER OPERATORS:

Operator	**Operator Name**	**Purpose**
*	Value at address Operator	Gives Value stored at Particular address
&	Address Operator	Gives Address of Variable

POINTER ADDRESS OPERATOR

1. Pointer address operator is denoted by '&' symbol
2. When we use ampersand symbol as a prefix to a variable name '&', it gives the address of that variable.

Take an example –

&n - It gives an address of variable n

WORKING OF ADDRESS OPERATOR

Examples:

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

```
int n = 10;
printf("\nValue of n is : %d",n);
printf("\nAddress of n is : %u",&n);
}
```

<u>Output :</u>

```
Value of  n is : 10
Address of n is : 1002
```

<u>Explanation:</u>

Consider the above example, where we have used to print the address of the variable using ampersand operator.

In order to print the variable we simply use name of variable while to print the address of the variable we use ampersand along with %u

printf("\nValue of &n is : %u",&n);

(2) #include<stdio.h>

```
Void main()
{
Int n=20;
Printf("The value of n is: %d",n);
Printf("The address of n is: %u",&n);
Printf("The value of n is: %d",*(&n));
}
```

<u>OUTPUT:</u>

```
The value of n is:20
The address of n is:1002
The value of n is:20
```

<u>Explanation:</u>

In the above program, first printf displays the value of n. The second printf displays the address of the variable n i.e) 1002, which is obtained by using &n(address of variable n). The last printf can be explained as follows,

```
                    *(&n) = *(Address of variable n)
                          =*(1002)
                          =Value at address 1002
          Therefore  *(&n)=20
```

UNDERSTANDING ADDRESS OPERATOR

Initialization of Pointer can be done using following 4 Steps :

i. Declare a Pointer Variable and Note down the Data Type.
ii. Declare another Variable with Same Data Type as that of Pointer Variable.
iii. Initialize Ordinary Variable and assign some value to it.
iv. Now Initialize pointer by assigning the address of ordinary variable to pointer variable.

Below example will clearly explain the initialization of Pointer Variable.

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

int a;       // Step 1
int *ptr;    // Step 2
a = 10;      // Step 3
ptr = &a;    // Step 4

return(0);
}
```

<u>Explanation of Above Program :</u>

- Pointer should not be used before initialization.
- “ptr” is pointer variable used to store the address of the variable.
- Stores address of the variable ‘a’ .
- Now “ptr” will contain the address of the variable “a” .

Note :
Pointers are always initialized before using it in the program

Consider the following program –

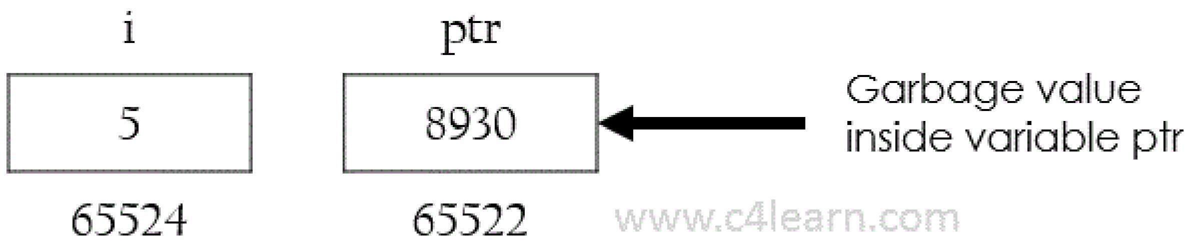
```
#include<stdio.h>
void main()
{
int i = 5;
int *ptr;
ptr = &i;
printf("\nAddress of i    : %u",&i);
printf("\nValue of ptr is : %u",ptr);
}
```

OUTPUT:

```
Address of i    : 65524
Value of ptr is : 65524
```

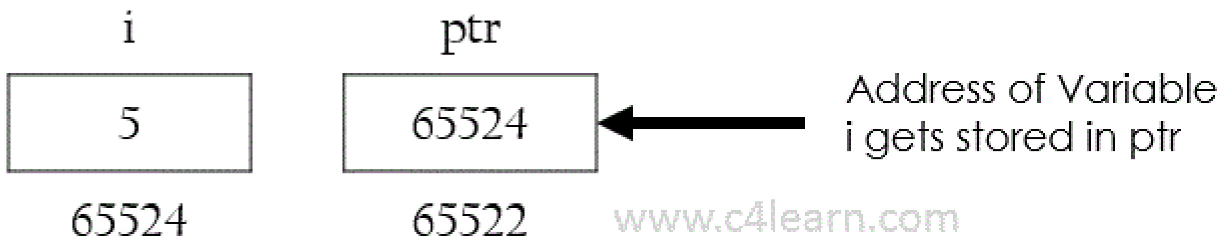
After declaration memory map will be like this –

```
int i = 5;
int *ptr;
```



After assigning the address of variable to pointer, i.e after the execution of this statement –

```
ptr = &i;
```



Program : accessing value and address of Pointer

```
/* Program to display the contents of the variable and their address using pointer variable*/
 (1)    #include<stdio.h>
        main()
        {
          int i = 3, *j;
          j = &i;
          printf("\nAddress of i = %u", &i);
          printf("\nAddress of i = %u", j);
          printf("\nAddress of j = %u", &j);
          printf("\nValue of j   = %u", j);
          printf("\nValue of i   = %d", i);
          printf("\nValue of i   = %d", *(&i));
          printf("\nValue of i   = %d", *j);
     }
```

Output :

```
Address of i = 65524
Address of i = 65524
Address of j = 65522
Value of   j = 65524
Value of   i = 3
Value of   i = 3
Value of   i = 3
```

Variable	Actual Value
Value of i	3
Value of j	65524
Address of i	65524
Address of j	65522

(2)

```
#include< stdio.h >
main()
{
int num, *intptr;
float x, *floptr;
char ch, *cptr;
num=123;
x=12.34;
ch='a';
intptr=&num;
cptr=&ch;
floptr=&x;
printf("Num %d stored at address %u\n",*intptr,intptr);
printf("Value %f stored at address %u\n",*floptr,floptr);
printf("Character %c stored at address %u\n",*cptr,cptr);
}
```

Output :

```
Num 123 stored at address  1000
Value 12.34 stored at address 2000
Character a stored at address 3000
```

POINTER EXPRESSIONS

Like any other variables pointer variables can be used in an expression. In general, expressions involving pointer conform to the same rules as other expressions. The pointer expression is a linear combination of pointer variables, variables and operators. Pointer expression gives either numerical output or address output.

Example:

```
        y = *p1 * *p2;
        sum = sum + *p1;
        z = 5 - *p2/*p1;
        *p2 = *p2 + 10;
```

```
/*Pointer expression and pointer arithmetic*/
#include< stdio.h >
void main()
{
int *ptr1,*ptr2;
int a,b,x,y;
a=30;b=6;
ptr1=&a;
ptr2=&b;
x=*ptr1+ *ptr2 –b;
y=b - *ptr1/ *ptr2 +a;
printf("\nAddress of a %u",ptr1);
printf("\nAddress of b %u",ptr2);
printf("\na=%d, b=%d",a,b);
printf("\nx=%d,y=%d",x,y);
}
```

OUTPUT:

```
Address of a 65522
Address of b 65524
a=30   b=6
x=30   y=31
```

EXPLANATION OF PROGRAM:

In the above example program, ptr1, ptr2 are the pointer variables which are used to store the address of the two variables a and b respectively using the statements ptr1=&a, ptr2=&b. In the pointer expressions which are given below, the value of x and y are calculated as follows,

```
x=*ptr1+ *ptr2 - b;
  =30 + 6 – 6
x=30
y=b - *ptr1/ *ptr2 +a;
  =6 - 30/6 + 30
  =6 – 5 + 30
y=31
```

POINTER ASSIGNMENT

We can use a pointer on the right-hand side of an assignment statement to assign its value to another pointer. For example,

```
#include< stdio.h >
void main()
{
int *p1,*p2;
int x=99;

p1=&x;
p2=p1;    /*pointer assignment*/
printf("\nValues at p1 and p2: %d %d",*p1,*p2);   /*print the value of x twice*/
printf("\nAddresses pointed to by p1 and p2: %u %u",p1,p2); /*print the address of x twice*/
}
```

OUTPUT:

```
 Values at p1 and p2: 99 99
 Addresses pointed to by p1 and p2: 5000 5000
```

EXPLANATION OF PROGRAM:

After the assignment sequence,

```
        p1=&x;
        p2=p1;
```

Both p1and p2 point to x. Thus both p1 and p2 refer to the same value.

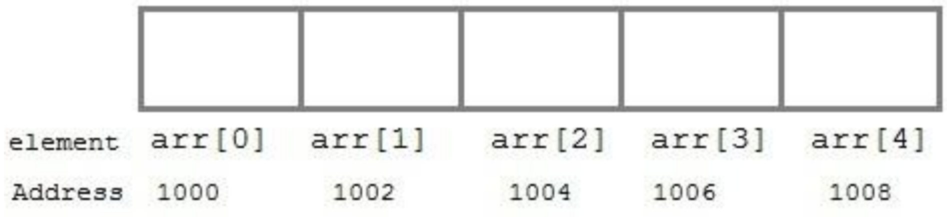
ARRAYS USING POINTER

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

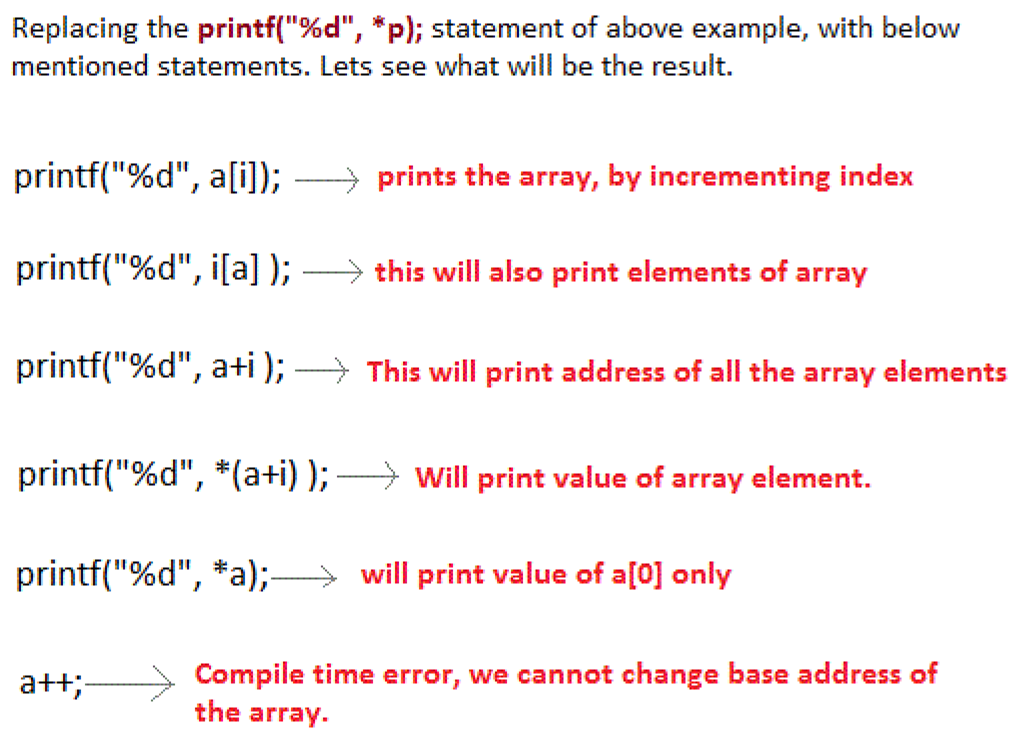
NOTE : You cannot decrement a pointer once incremented. p-- won't work.

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.



Replacing the **printf("%d", *p);** statement of above example, with below mentioned statements. Lets see what will be the result.

printf("%d", a[i]); $\longrightarrow$ **prints the array, by incrementing index**

printf("%d", i[a]); $\longrightarrow$ **this will also print elements of array**

printf("%d", a+i); $\longrightarrow$ **This will print address of all the array elements**

printf("%d", *(a+i)); $\longrightarrow$ **Will print value of array element.**

printf("%d", *a); $\longrightarrow$ **will print value of a[0] only**

a++; $\longrightarrow$ **Compile time error, we cannot change base address of the array.**

/*Program to print the addresses of array elements */

```
#include <stdio.h>
void main(){
   char c[4];
   int i;
   for(i=0;i<4;++i){
      printf("Address of c[%d]=%x\n",i,&c[i]);
   }
}
```

<u>OUTPUT:</u>

```
Address of c[0]=28ff44
Address of c[1]=28ff45
Address of c[2]=28ff46
Address of c[3]=28ff47
```

Notice, that there is equal difference (difference of 1 byte) between any two consecutive elements of array.

Consider the following:

```
int my_array[] = {1,23,17,4,-5,100};
```

Here we have an array containing 6 integers. We refer to each of these integers by means of a subscript to my_array, i.e. using my_array[0] through my_array[5]. But, we could alternatively access them via a pointer as follows:

```
int *ptr;
ptr = &my_array[0];       /* pointer points to the first integer in our array */
```

And then we could print out our array either using the array notation or by dereferencing our pointer. The following code illustrates this:

```
#include <stdio.h>
int main(void)
 {
int my_array[] = {1,23,17,4,-5,100};
 int *ptr;
 int i;
 ptr = &my_array[0] ;  /* point pointing to the first element of the array */
 printf("\n\n");
for (i = 0; i < 6; i++)
{
 printf("my_array[%d] = %d ",i,my_array[i]);          /*<-- A */
printf("ptr + %d = %d\n",i, *(ptr + i));               /*<-- B */
 }
 return 0;
}
```

Compile and run the above program and carefully note lines A and B and that the program prints out the same values in either case. Also observe how we dereferenced our pointer in line B, i.e. we first added i to it and then dereferenced the new pointer. Change line B to read:

```
printf("ptr + %d = %d\n",i, *ptr++);
```

and run it again. then change it to:

```
printf("ptr + %d = %d\n",i, *(++ptr));
```

and try once more.Each time try and predict the outcome and carefully look at the actual outcome.

In C, the standard states that wherever we might use &var_name[0] we can replace that with var_name, thus in our code where we wrote:

```
ptr = &my_array[0];
```

we can write:

```
ptr = my_array;
```

to achieve the same result.

/* Example program to print the array elements using pointer */

```
#include <stdio.h>
int main(){
   int data[5], i;
   printf("Enter elements: ");
   for(i=0;i<5;++i)
     scanf("%d",data[i]);
   printf("You entered: ");
   for(i=0;i<5;++i)
      printf("%d\n",*(data+i));
   return 0;
}
```

Output

```
Enter elements: 1
2
3
5
4
You entered: 1
2
3
5
4
```

/* Program to find sum of array elements using pointer */

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int numArray[10];
   int i, sum = 0;
   int *ptr;

   printf("\nEnter 10 elements : ");

   for (i = 0; i < 10; i++)
      scanf("%d", &numArray[i]);

   ptr = numArray;

   for (i = 0; i < 10; i++) {
      sum = sum + *ptr;
      ptr++;
   }
 printf("The  sum  of  array  elements  :
%d", sum);
}
```

OUTPUT

```
Enter 10 elements : 11 12 13 14 15 16 17 18 19 20
The sum of array elements is 155
```

EXPLANATION OF PROGRAM:

Accept the 10 elements from the user in the array.

```
for (i = 0; i < 10; i++)
 scanf("%d", &numArray[i]);
```

We are storing the address of the array into the pointer.

```
ptr = numArray;            /* a=&a[0] */
```

Now in the for loop we are fetching the value from the location pointer by pointer variable. Using De-referencing pointer we are able to get the value at address.

```
for (i = 0; i < 10; i++) {
   sum = sum + *ptr;
   ptr++;
}
```

Suppose we have 2000 as starting address of the array. Then in the first loop we are fetching the value at 2000. i.e

```
sum = sum + (value at 2000)
    = 0   + 11
    = 11
```

In the Second iteration we will have following calculation –

```
sum = sum + (value at 2002)
    = 11  + 12
    = 23
```

Pointer example-1

```
#include <stdio.h>
#include <math.h>
main()
{
            int num [ ] = { 10,20,30,40,50 }
            print ( &num, 5, num);
}
print ( int *j, int n, int b[5])
{
            int i;
            for(i=0;i<=4;i++)
```

```
        {
                printf (  "  %u  %d  %d  %u  \n  ", &j[i]  ,  *j , *(b+i)  ,  &b);
                j++;
        }
}
```

In this example we have a single dimensional array num and a function print . We are passing, the address to the first element of the array, the number of elements and the array itself, to this function. When the function receives this arguments, it maps the first one to another pointer j and the array num is copied into another array b . (The type declarations are made here itself. Note that these declarations can also be given just below this line). j is now a pointer to the array b.

Inside the function we are printing out the address of the array element and the value of the array element in two ways. One using the pointer j and the other using the array b. If we compile and run this code we get the following out put,

```
3221223408 10 10 3221223376
3221223416 20 20 3221223376
3221223424 30 30 3221223376
3221223432 40 40 3221223376
3221223440 50 50 3221223376
```

Note that as we increment j it points to the successive elements of the array. We can get both the address of the array elements and the value stored there using this. However the array name, which acts also as the pointer to its base address, is not able to give us the address of its elements. Or in other words, the array name is a constant pointer. Also note that while j ispoints to the elements of the array num, b is pointing to its copy.

Next we have an example that uses a two dimensional array. Here care should be taken to declare the number of columns correctly.

Pointer example-2

```
#include <stdio.h>
#include <math.h>
main()
{
        int arr [ ][3] = {{11,12,13}, {21,22,23},{31,32,33},{41,42,43},{51,52,53}};
        int I , j ;
        int  *p ,  (*q) [3], *r ;
        p = (int *) arr ;
        q = arr;
        r = (int *) q ;
        printf (  "  %u  %u  %d  %d  %d  %d \n  ", p ,  q  ,  *p , *(r)  ,  *(r+1),  *(r+2));
        p++ ;
        q++ ;
        r = (int *) q ;
        printf (  "  %u  %u  %d  %d  %d  %d \n  ", p ,  q  ,  *p , *(r)  ,  *(r+1),  *(r+2));


}
```

Here we have a pointer p and a pointer array q. The first assignment statement is to make the pointer p points to the array arr. While assigning, we also declare the type of the variable arr. Note that variables on both side of this statement should have the same type. Next line is a similar statement, now with q and arr. Since q is a pointer array, the array can be directly assigned to it and there is no need for specifying the type of the variable. In the next line we make the pointer r to point to the pointer array q . Then we will print out the different values. Here is what we get from this,

```
        3221223344  3221223344  11 11 12 13
        3221223348  3221223356  12 21 22 23
```

Here we see that incrementing p make it just jump through each element of the array, where as incrementing q, will move it from one row to another row.

ARRAY OF POINTERS

Just like array of integers or characters, there can be array of pointers too. An array of pointers can be declared as :

<datatype> *<pointername> [number-of-elements];

For example :

```
        char *ptr[3];
```

The above line declares an array of three character pointers. Let’s take a working example :

```
#include<stdio.h>
int main(void)
{
    char *p1 = "Himanshu";
    char *p2 = "Arora";
    char *p3 = "India";

    char *arr[3];

    arr[0] = p1;
    arr[1] = p2;
    arr[2] = p3;

   printf("\n p1 = [%s] \n",p1);
   printf("\n p2 = [%s] \n",p2);
   printf("\n p3 = [%s] \n",p3);

   printf("\n arr[0] = [%s] \n",arr[0]);
   printf("\n arr[1] = [%s] \n",arr[1]);
   printf("\n arr[2] = [%s] \n",arr[2]);

   return 0;
}
```

In the above code, we took three pointers pointing to three strings. Then we declared an array that can contain three pointers. We assigned the pointers 'p1′, 'p2′ and 'p3′ to the 0,1 and 2 index of array.

Let's see the output :

```

 p1 = [Himanshu]
 p2 = [Arora]
 p3 = [India]
 arr[0] = [Himanshu]
 arr[1] = [Arora]
 arr[2] = [India]
```

So we see that array now holds the address of strings.

Let us consider the following example, which makes use of an array of 3 integers:

```
int main ()
{
   int  var[] = {10, 100, 200};
   int i;

   for (i = 0; i < 3; i++)
   {
      printf("Value of var[%d] = %d\n", i, var[i] );
   }
   return 0;
}
```

<u>OUTPUT:</u>

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer:

```
                              int *ptr[3];
```

This declares ptr as an array of 3 integer pointers. Thus, each element in ptr, now holds a pointer to an int value.

Following example makes use of three integers, which will be stored in an array of pointers as follows:

```
#include <stdio.h>
 int main ()
{
   int  var[] = {10, 100, 200};
   int i, *ptr[3];

      ptr[0] = &var[0];  /* assign the address of 1st integer element */
      ptr[1] = &var[1];  /* assign the address of 2nd integer element */
      ptr[2] = &var[2];  /* assign the address of 3rd integer element */

   for ( i = 0; i < 3; i++)
   {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
   }
   return 0;
}
```

<u>OUTPUT:</u>

```
 Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

Questions to practice:

1. Write a program to find largest element of an array using pointer.
2. Write a program to calculate average of array elements using pointer.
3. Write a program to sort elements of an array using pointer.
4. Write a program to print the value and address of each element of an array using pointer.
5. What is the output of the following program?

```
#include<stdio.h>
int main()
{
int i;
char *arr[4] = {"C","C++","Java","VBA"};
char *(*ptr)[4] = &arr;
for(i=0;i<4;i++)
 printf("Address of String %d : %u\n",i+1,(*ptr)[i]);
return 0;}
```

6. What is the output of the following program?

```
#include<stdio.h>
int main()
{
int i;
char *arr[4] = {"C","C++","Java","VBA"};
char *(*ptr)[4] = &arr;
printf("%s",++(*ptr)[2]);
return 0;
}
```

7. Consider the following code and find the output.

```
main()
{
int x[] = {1,2,3,4,5};
int *ptr,i ;
ptr = x
for(i=0;i<5;i++)
   {
   printf("nAddress : %u",&x[i]);
   printf("nElement : %d",x[i]);
   printf("nElement : %u",*(x+i));
   printf("nElement : %d",i[x]);
   printf("nElement : %d",*ptr);
   }
}
```

8. How much Memory required to store Pointer variable?

9. What is the output of the following program?

```
#include<stdio.h>
int main()
{
int a = 10, *ptr;
char a = 'a', *cptr;
float a = 3.14, *fptr;
ptr = &a;
cptr = &a;
fptr = &a;
printf("\nSize of Integer Pointer : %d",sizeof(ptr));
printf("\nSize of Character Pointer : %d",sizeof(cptr));
printf("\nSize of Character Pointer : %d",sizeof(fptr));
return(0);
}
```

10. Are *ptr++ and ++*ptr are same ? Justify your answer.

11. Give the output of following programs.

(i) Incrementing Integer Pointer

```
#include<stdio.h>
int main(){
int *ptr=(int *)1000;
ptr=ptr+1;
printf("New Value of ptr : %u",ptr);
return 0;}
```

(ii) Program to Compute Difference between Pointers:

```
#include<stdio.h>
int main()
{
int num , *ptr1 ,*ptr2 ;
ptr1 = &num ;
ptr2 = ptr1 + 2 ;
printf("%d",ptr2 - ptr1);
return(0);
}
```

(iii) Comparison between two Pointers:

```
#include<stdio.h>
int main()
{
int *ptr1,*ptr2;
ptr1 = (int *)1000;
ptr2 = (int *)2000;
if(ptr2 > ptr1)
   printf("Ptr2 is far from ptr1");
return(0);
}
```

Pointer and Function

Function Pointer

```
#include <stdio.h>
 void subtractAndPrint(int x, int y);
 void subtractAndPrint(int x, int y) {
  int z = x - y;
  printf("Simon says, the answer is: %d\n", z);
}
 int main()
 {
  void (*sapPtr)(int, int) = subtractAndPrint;
  (*sapPtr)(10, 2);
  sapPtr(10, 2);
}
```

The pointer can be used as an argument in functions. The arguments or parameters to the function are passed in two ways.

- ❖ Call by value
- ❖ Call by reference
- ➢ In ‘C Language there are two ways that the parameter can be passed to a function they are
  - o **Call by value**
  - o **Call by reference**

Call by Value:

- ➢ This method copies the value of actual parameter into the formal parameter of the function.
- ➢ The changes of the formal parameters cannot affect the actual parameters, because formal arguments are photocopy of the actual argument.
- ➢ The changes made in formal argument are local to the block of the called functions. Once control return back to the calling function the changes made disappear.

Example:

```
#include<stdio.h>
#include<conio.h>
void cube(int);
int cube1(int);
void main()
{
        int a;
        clrscr();
        printf(“Enter one values”);
        scanf(“%d”,&a);
        printf(“Value of cube function is=%d”, cube(a));
        printf(“Value of cube1 function is =%d”, cube1(a ));
        getch();
}
void cube(int x)
{
        x=x*x*x*;
        return x;
}

int cube1(int x)
{
        x=x*x*x*;
        return x;
}
```

Output:

```
Enter one values  3
Value of cube function is  3
Value of cube1 function is 729
```

Call by reference

- Call by reference is another way of passing parameter to the function.
- Here the address of argument are copied into the parameter inside the function, the address is used to access arguments used in the call.
- Hence changes made in the arguments are permanent.
- Here pointer are passed to function, just like any other arguments.

Example:-

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
        int a=5,b=10;
        clrscr();
        printf("Before swapping a=%d b=%d",a,b);
        swap(&a,&b);
        printf("After swapping a=%d b=%d",a,b);
        getch();
}
void swap(int *x,int *y)
{
        int *t;
        t=*x;
        *x=*y;
        *y=t;
}
```

Output:

```
Before swapping a=5  b=10
After swapping a=10  b=5
```

Function Returning Pointer

A function can return a single value by its name or return multiple values through pointer parameters. Since pointers are a data type in c, we can also force a function to return a pointer to the calling function.

Program:

```
int *larger(int*,int*);
void main()
{
        int a=10;
        int  b=20;
        int *p;
        p =larger(&a,&b);
        printf("%d",*p);
}
int *larger(int *x, int *y)
```

Output:

```
20
```

```
{
        if(*x>*y)
                return(x);
        else
                return (y);
}
```

STRUCTURE USING POINTER

```
struct name
{
   member1;
   member2;
   .
   .
};
-------- Inside function -------
struct name *ptr;
```

Here, the pointer variable of type **struct name** is created.

Structure's member through pointer can be used in two ways:

1. Referencing pointer to another address to access memory
2. Using dynamic memory allocation

Consider an example to access structure's member through pointer.

```
#include <stdio.h>
struct name{
  int a;
  float b;
};
int main()
{
   struct name *ptr,p;
   ptr=&p;          /* Referencing pointer to memory address of p */
   printf("Enter integer: ");
   scanf("%d",&(*ptr).a);
   printf("Enter number: ");
   scanf("%f",&(*ptr).b);
   printf("Displaying: ");
   printf("%d%f",(*ptr).a,(*ptr).b);
   return 0;
}
```

In this example, the pointer variable of type **struct name** is referenced to the address of *p*. Then, only the structure member through pointer can can accessed.

Structure pointer member can also be accessed using -> operator.

(*ptr).a is same as ptr->a

(*ptr).b is same as ptr->b

ACCESSING STRUCTURE MEMBERS WITH POINTER

To access members of structure with structure variable, we used the dot `.` operator. But when we have a pointer of structure type, we use arrow `->` to access structure members.

```
struct Book
{
 char name[10];

 int price;

}



int main()

{

 struct Book b;

 struct Book* ptr = &b;

 ptr->name = "Dan Brown";     //Accessing Structure Members

 ptr->price = 500;

}
```

Example program for C structure using pointer:

In this program, “record1″ is normal structure variable and “ptr” is pointer structure variable. As you know, Dot(.) operator is used to access the data using normal structure variable and arrow(->) is used to access data using pointer variable.

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

struct student
{
     int id;
     char name[30];
     float percentage;
};

int main()
{
     int i;
     struct student record1 = {1, "Raju", 90.5};
     struct student *ptr;

     ptr = &record1;

         printf("Records of STUDENT1: \n");
         printf("  Id is: %d \n", ptr->id);
         printf("  Name is: %s \n", ptr->name);
         printf("  Percentage is: %f \n\n", ptr->percentage);

     return 0;
}
```

Output:

```
Records of STUDENT1:
Id is: 1
Name is: Raju
Percentage is: 90.500000
```