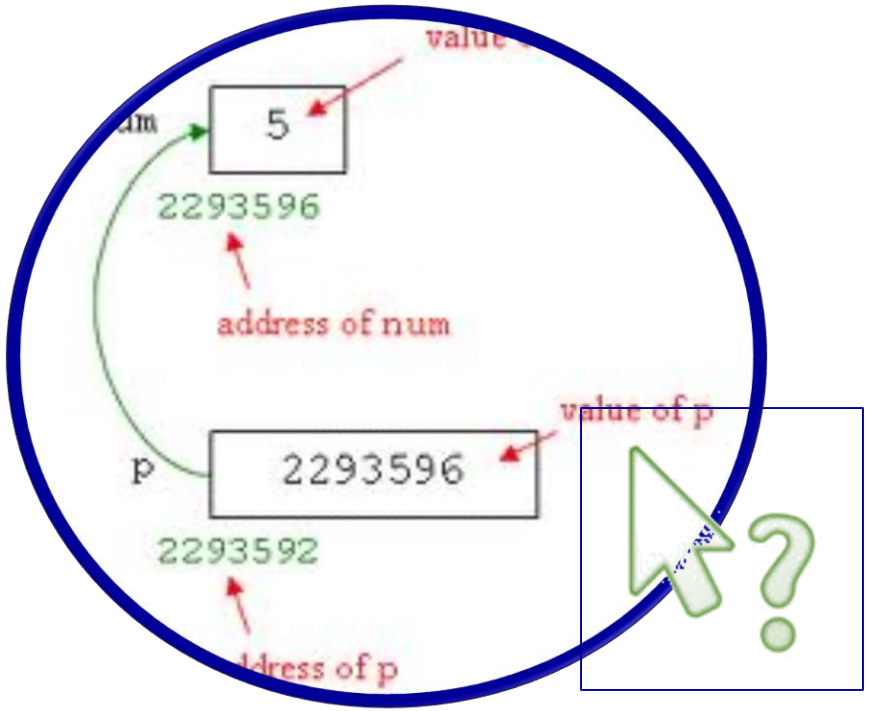
S23_1 Pointers



Pointers

Objectives

To learn and appreciate the following concepts

- Pointers and Arrays
- Pointers and Character Strings
- Pointers and 2D
- Array of Pointers

Session outcome

At the end of session one will be able to understand

- Basic operations on pointers
- Pointers and Arrays
- Pointers and Character Strings
- Pointers and 2D
- Array of Pointers

Pointers and arrays

- When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.

- The base address is the location of the first element (index 0) of the array.

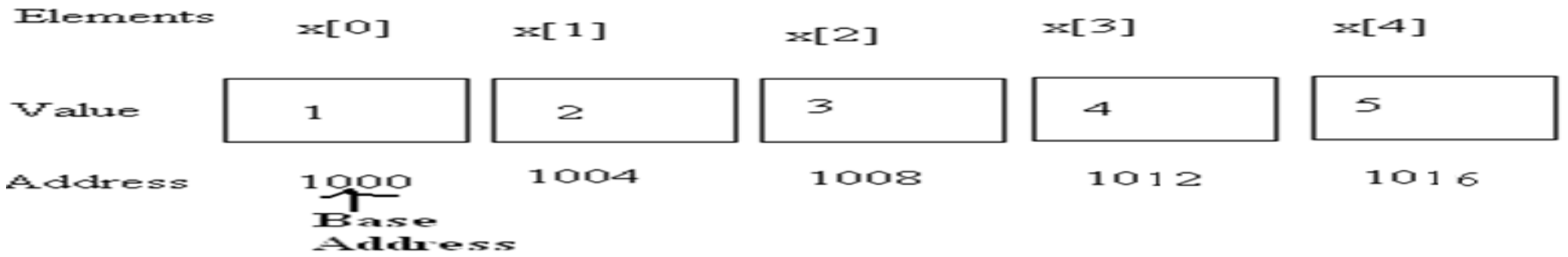
- The compiler also defines the array name as a **constant pointer** to the first element.

Pointers and arrays

- An array x is declared as follows and assume the base address of **x** is **1000.**

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

- Array name **x**, is a constant pointer, pointing to the first element **x[0]** .
- Value of **x** is **1000 (Base Address)**, the location of **x[0]**. i.e. **x = &x[0] = 1000 (in the example below)**



Array accessing using Pointers

- An **integer pointer variable** p, can be made to point to an array as follows:

```
int  x[5] ={ 1,2,3,4,5};
int *p;
p = x;     OR      p = &x[0];
```

- **Following statement is Invalid:**

```
p = &x ; //Invalid
```

- **Successive array elements can be accessed** by writing:

```
printf("%d", *p); p++;
or
printf("%d", *(p+i)); i++;
```

Pointers and arrays

- The relationship between **p** and **x** is shown below:

```
p= &x[0];        (=1000) BASE ADDRESS
p+1=>&x[1]       (=1004)
p+2=>&x[2]       (=1008)
p+3=>&x[3]       (=1012)
p+4=>&x[4]       (=1016)
```

- **Address of an element of x is given by:**

Address of **x[i]** = **base address** + **i** * **scale factor of (int)**

Address of x[3]= 1000 +(3*4) = 1012

Array accessing using array name as pointer - Example

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

 int arr[5] = { 31, 54, 77, 52, 93 };

 for(int j=0; j<5; j++)        //for each element,
 printf("%d ", *(arr+j));              //print value

 return 0;
}
```

```
31 54 77 52 93
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Array accessing using Pointers - Example

```
// array accessed with pointer
#include <stdio.h>

int main()
{

  int arr[5] = { 31, 54, 77, 52, 93 };
  int* ptr;          //pointer to arr
  ptr = arr;   //points to arr

 for(int j=0; j<5; j++)     //for each element,
 printf("%d ", *ptr++);

return 0;
}
```

```
31 54 77 52 93
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

“ptr” is a pointer which can be used to access the elements.

Sum of all elements stored in an array

```
#include <stdio.h>
int main()
{
   int  *p, sum=0, i=0;
   int  x[5] ={5, 9, 6,3,7};
   p=x;
   while(i<5)
   {
     sum+=*p;
     i++;
     p++;
   }
   printf("sum of elements = %d“, sum);
   return 0;
```

```
sum of elements =30
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Pointers & Character strings

```
//length of the string
#include <stdio.h>
int main()
{
  char name[15];
  char *cptr=name;
  printf("Enter some word to find its length: \n“);
  scanf(“%s”, name);
  while(*cptr!= '\0')
          cptr++;
  printf("length= %d“,cptr-name);
  return 0;
}
```

```
Enter some word to find its length
Computer
length=8
Process returned 0 (0x0)   execution time : 16.345 s
Press any key to continue.
```

Pointers & Character strings

- The statements

```
char name[10];
char  *cptr =name;
```

declares cptr as a pointer to a character array and assigns address of the first character of name as the initial value.

- The statement while(*cptr!='\0')

is true until the end of the string is reached.

- When the while loop is terminated, the pointer cptr holds the address of the null character ['\0'].
- The statement length = cptr – name; gives the length of the string name.

Pointers & Character strings

- A constant character string always represents a pointer to that string.
- The following statements are valid.

```
char *name;

name =“Delhi”;
```

These statements will declare **name** as a **pointer** to character array and assign to name the constant character string “Delhi”.

Pointers and 2D arrays

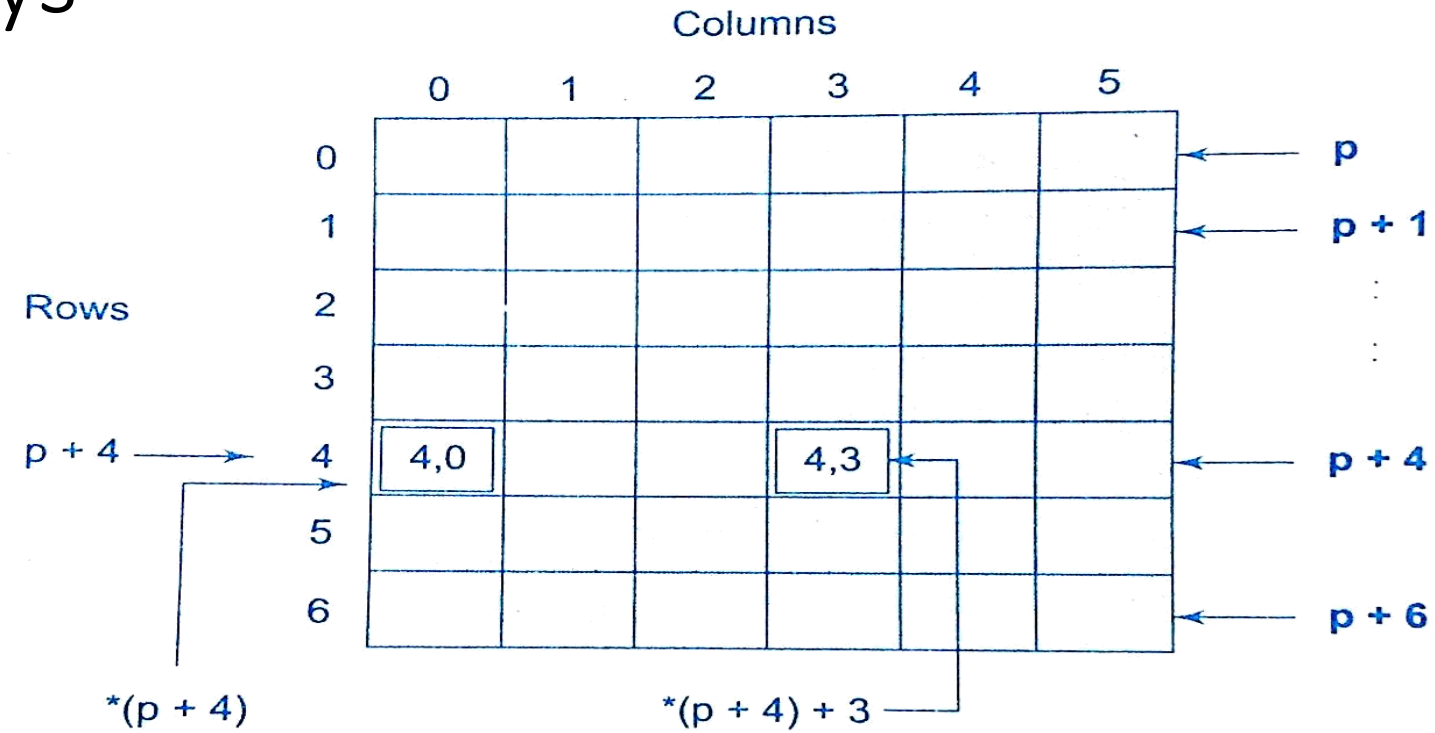
```
int a[][2]={  {12, 22},
              {33, 44} };
int  (*p)[2];
p=a; // initialization
```

Element in 2d represented as

***(*(a+i)+j)**

or

***(*(p+i)+j)**



Pointers and 2D arrays

```
// 2D array accessed with pointer
#include <stdio.h>
int main()
{

   int i, j, (*p)[2], a[][2] = {{12, 22}, {33, 44} };

   p=a;

   for(i=0;i<2;i++)
   {
      for(j=0;j<2;j++)
         printf(“%d \t“, *(*(p+i)+j));
      printf("\n“);
   }

   return 0;
}
```

```
12      22
33      44

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Array of pointers

- We can use pointers to handle a table of strings.

  char name[3][25];

  name is a table containing 3 names, each with a maximum length of 25 characters (including '\0')

- Total **storage** requirement for **name** is **75 bytes**.

  But rarely all the individual strings will be equal in lengths.

- We can use a pointer to a string of varying length as

char *name[3] = { "New Zealand", "Austrailia", "India" };

Array of pointers

So, **char *name[3] = { “New Zealand “,**

“Australia”,

“India”};

Declares **name** to be an **array of 3 pointers** to characters, each pointer pointing to a particular name.

name[0] → New Zealand

name[1]→ Australia

name[2]→ India

This declaration allocates 28 bytes.

Array of pointers

The following statement would print out all the 3 names.

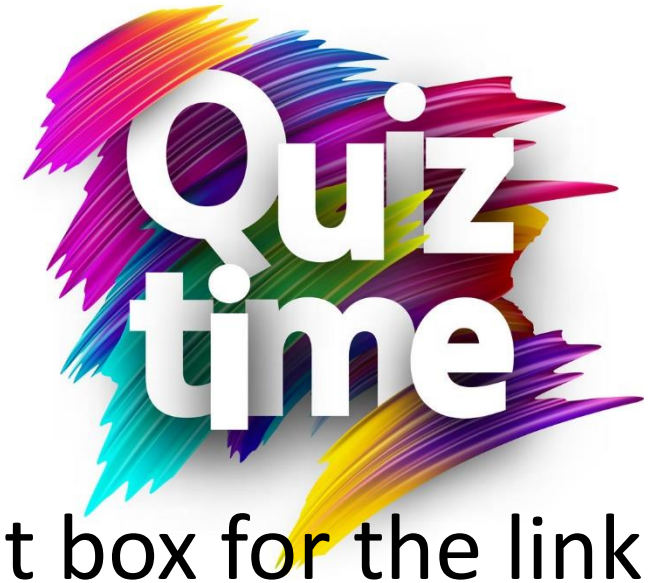
```
for(i=0; i<=2;i++)
  printf("%s",name[i]);
```

or `printf("%s", *(name + i));`

To access the j^{th} character in the i^{th} name, we may write as

`*(name[i] +j)`

The character array with rows of varying lengths are called **ragged arrays** and are better handled by pointers.



Go to posts/chat box for the link to the question

submit your solution in next 2 minutes

The session will resume in 3 minutes

Summary of pointers

- Pointers and Arrays
- Pointers and Strings
- Array of Pointers