SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A’ Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit IV – FUNCTIONS AND POINTERS

Topic : Pointers and Arrays

Topics Covered

- **Pointers:**
  - **Pointers and Arrays**
  - **Pointers and character Strings**
  - **Array of Pointers**
  - **Function Returning 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 array in contiguous memory locations.

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

Pointers and Arrays

- Suppose the base address of x is 1000 and assuming that each integer requires two bytes, the five elements will be stored as follows:

Elements	x[0]	x[1]	x[2]	x[3]	x[4]
Value	**1**	**2**	**3**	**4**	**5**
Address	1000	1002	1004	1006	1008

Pointers and Arrays

- The name x is defined as a constant pointer pointing to the first element x[0] and therefore value of x is 1000, the location where x[0] is stored .

```
int p = x;
p = &x[0]=1000;
```

- Here p is an integer pointer.
- We can access the value of x using p++ (Pointer variable with increment operator) to move from one element to another.

Pointers and Arrays

- **Example:**

  p = &x[0] (=1000)

  p+1 = &x[1] (=1002)

  ….

  p+4 = &x[4] (=1008)

- **Address of element is calculated using its index and the scale factor of the data type**.

- Address of x[3] = base address + (3 x scale factor of int)

  =1000 + (3 x 2) =1006

Pointers and Arrays

Pointer to access one-dimensional array elements:

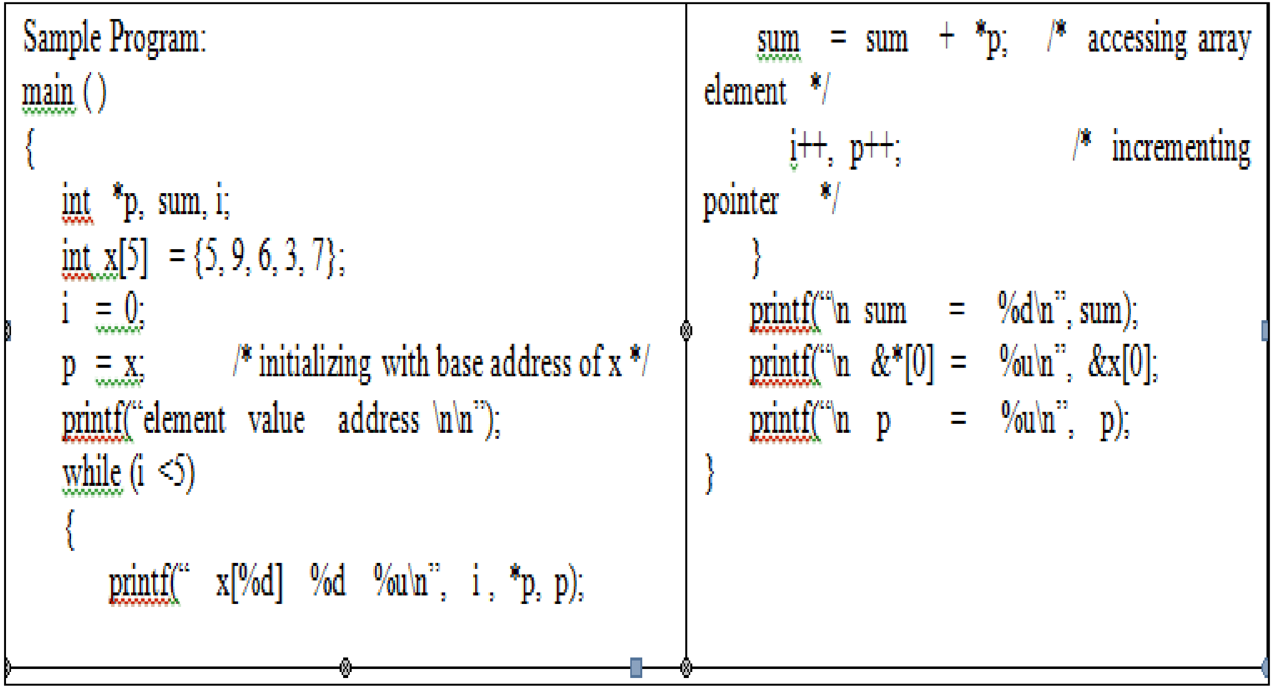
- We can use pointers to access array elements.

  Note that *(p+3) gives the value of x[3].

- The pointer accessing method is much faster than array indexing.

- Similarly pointers can be used to manipulate two-dimensional arrays.

Pointers and Arrays



Sample Program:

```
main ( )
{
    int  *p, sum, i;
    int x[5]  = {5, 9, 6, 3, 7};
    i  = 0;
    p  = x;          /* initializing with base address of x */
    printf(“element   value    address  \n\n”);
    while (i <5)
    {
        printf(“   x[%d]   %d   %u\n”,  i , *p, p);
        sum   =  sum   +  *p;    /* accessing array
element  */
          i++,  p++;                  /* incrementing
pointer   */
    }
    printf(“\n sum     =   %d\n”, sum);
    printf(“\n  &*[0]  =   %u\n”,  &x[0];
    printf(“\n  p      =   %u\n”,   p);
}
```

Pointers and Arrays

```
#include <stdio.h>
int main()
{
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++;
}
return 0;
}
```

Pointers and Character Strings

- Strings are treated like **character arrays** and therefore, they are declared and initialized as follows:

char str [5] = “good”;

- The **compiler automatically inserts the null character ‘\0’ at the end of the string.**

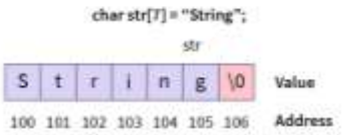
- C supports an alternative method to create strings using pointer variables of type char.

Pointers and Character Strings

Example:

char * str= “String”;

- This creates a string for the literal and then stores its address in the pointer variable str.
- The pointer str now points to the first character of the string “String” as:



char str[7] = "String";

str

S	t	r	i	n	g	\0	Value
100	101	102	103	104	105	106	Address

- We can also use the runtime assignment for giving values to a string pointer

Pointers and Character Strings

```
char  *  string1;

string1  = “good”;
```

- Note that the assignment,

```
string1  = “good”;
```

- is not a string copy, because the **variable string1 is a pointer, not a string**

Pointers and Character Strings

- A string is a sequence of characters which we save in an array.
- And in C programming language the \0 null character marks the end of a string.

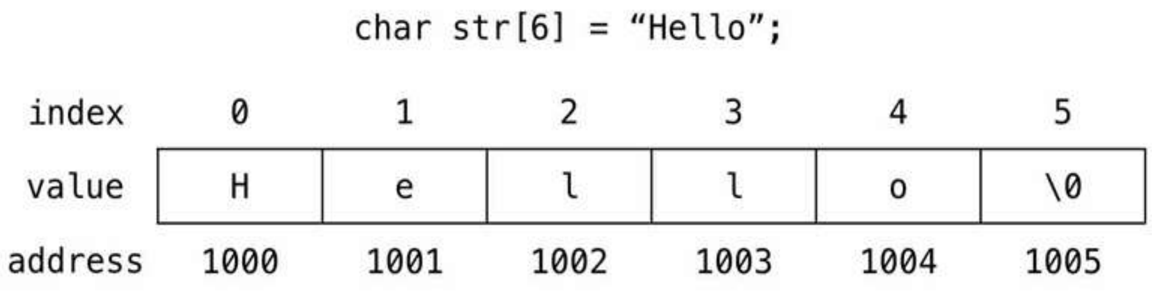
Creating a string:

- In the following example we are creating a string str using char character array of size 6.

```
char str[6] = "Hello";
```

Pointers and Character Strings

- The above string can be represented in memory as follows.
- Each character in the string str takes 1 byte of memory space.



```
char str[6] = “Hello”;
```

index	0	1	2	3	4	5
value	H	e	l	l	o	\0
address	1000	1001	1002	1003	1004	1005

Pointers and Character Strings

- In the following code we are assigning the address of the string str to the pointer ptr.

```
char *ptr = str;
```

Accessing string via pointer:

- To access and print the elements of the string we can use a loop and check for the \0 null character.

Pointers and Character Strings

```
main( )
 {
        char *name;
        int length;
     char *cptr=name;
        name=”delhi”;
        printf(“%s”,name);
        while (cptr != ‘\0’)
        {
                     printf(“%c is stored at address %u \n”,*cptr,cptr);
                     cptr++;
        }
        length=cptr-name;
        printf(“%d”,length);
}
```

```
#include <stdio.h>
int main( )
{
char str[6] = "Hello"; // string variable
char *ptr = str; // pointer variable
while(*ptr != '\0') // print the string
{
printf("%c", *ptr);
// move the ptr pointer to the next memory location
ptr++;
}
return 0;
}
```

Array of Pointers

Pointer to an array is also known as array pointer. We are using the pointer to access the components of the array.

```
int a[3] = {3, 4, 5 };
int *ptr = a;
```

We have a pointer ptr that focuses to the 0th component of the array. Similarly a pointer can be declared that point to whole array rather than just a single component of the array.

Array of Pointers

Syntax:

data type (*var name)[size of array];

// pointer to an array of five numbers int (* ptr)[5] = NULL;

The above declaration is the pointer to an array of five integers. We use parenthesis to pronounce pointer to an array.

Since subscript has higher priority than indirection, it is crucial to encase the indirection operator and pointer name inside brackets.

Array of Pointers

```
#include <stdio.h>
 int main()
{
     // Pointer to an array of five numbers
   int(*a)[5];
     int b[5] = { 1, 2, 3, 4, 5 };
     int i = 0;
     // Points to the whole array b
     a = &b;
     for (i = 0; i < 5; i++)
     printf("%d\n", *(*a + i));
     return 0;
}
```

Array of Pointers

“Array of pointers” is an array of the pointer variables. It is also known as pointer arrays.

Syntax:

```
int *var_name[array_size];
```

Declaration of an array of pointers:

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

We can make separate pointer variables which can point to the different values or we can make one integer array of pointers that can point to all the values.

Array of Pointers

```
// C program to demonstrate // example of array of pointers.
 #include <stdio.h>
 const int SIZE = 3;
 void main()
{
   // creating an array
  int arr[] = { 1, 2, 3 };
   // we can make an integer pointer array to
  // storing the address of array elements
  int i, *ptr[SIZE];
   for (i = 0; i < SIZE; i++) {
     // assigning the address of integer.
    ptr[i] = &arr[i];
  }
   // printing values using pointer
  for (i = 0; i < SIZE; i++) {
     printf("Value of arr[%d] = %d\n", i, *ptr[i]);
  }
}
```

Array of Pointers

```
#include<stdio.h>
  const int size = 4;
 void main()
{

    // array of pointers to a character
    // to store a list of strings
    char* names[] = {
        "amit",
        "amar",
        "ankit",
        "akhil"
    };
      int i = 0;
      for (i = 0; i < size; i++) {
        printf("%s\n", names[i]);
    }
}
```

Array of Pointers

```
// C program to understand difference between pointer to an integer and pointer to an array of integers.
#include<stdio.h>
int main()
{
// Pointer to an integer
int *p;
// Pointer to an array of 5 integers
int (*ptr)[5];
int arr[5];
// Points to 0th element of the arr.
p = arr;
// Points to the whole array arr.
ptr = &arr;
printf("p = %p, ptr = %p\n", p, ptr);
p++;
ptr++;
printf("p = %p, ptr = %p\n", p, ptr);
return 0;
}
```

Array of Pointers

p: is pointer to 0th element of the array *arr*, while ***ptr*** is a pointer that points to the whole array *arr*.

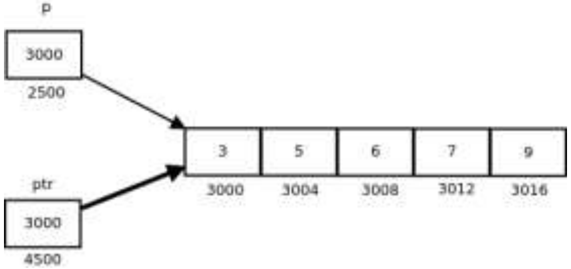
The base type of *p* is int while base type of *ptr* is ‘an array of 5 integers’.

We know that the pointer arithmetic is performed relative to the base size, so if we write ptr++, then the pointer *ptr* will be shifted forward by 20 bytes.

On dereferencing a pointer expression we get a value pointed to by that pointer expression.

Array of Pointers

- Pointer to an array points to an array, so on dereferencing it, we should get the array, and the name of array denotes the base address.
- So whenever a pointer to an array is dereferenced, we get the base address of the array to which it points.
- The following figure shows the pointer p and ptr. Darker arrow denotes pointer to an array.



Function Returning Pointers

- C also allows to return a pointer from a function.
- We can pass pointers to the function as well as return pointer from a function.
- But it is not recommended to return the address of a local variable outside the function as it goes out of scope after function returns.

Function Returning Pointers

Syntax

```
return_type  *function_name(parameter_list)
{

   // function body

 }
```

Function Returning Pointers

Examples

```
          int *func(int, int);
           // this function returns a pointer to
int

          double *func(int, int);
          // this function returns a pointer to
double
```

Function Returning Pointers

```
#include <stdio.h>
int *returnPointer(int *p);
int main()
{
int i=10;
int *ptr1, *ptr2;
ptr1=&i;
ptr2=returnPointer(&i);
printf("\n *ptr1 = %d",*ptr1);
printf("\n *ptr2 = %d",*ptr2);
return 0;
}
int *returnPointer(int *pt)
{
return pt;   }
```

Function Returning Pointers

```
#include<stdio.h>
int *return_pointer(int *, int); // this function returns a pointer of type int
int main()
{
int i, *ptr;
int arr[] = {11, 22, 33, 44, 55};
i = 4;
printf("Address of arr = %p\n", arr);
ptr = return_pointer(arr, i);
printf("\nAfter incrementing arr by 4 \n\n");
printf("Address of ptr = %p\n\n", ptr);
printf("Value at %p is %d\n", ptr, *ptr);
// signal to operating system program ran fine
return 0;}
int *return_pointer(int *p, int n)
{
p = p + n;
return p;
}
```

Summary

- Pointer variables can be used in expressions.
- We may also use short-hand operators with the pointers.
- Pointer can also be compared using the relational operators.
- When the pointers are used for character array or strings, then it is called as string pointers.
- We can create a pointer to store the address of an array. This created pointer is called a pointer to an array also known as an array pointer.

Thank
you