VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102L- Structured and object-oriented programming

Module 3

Dr. P.Keerthika

Associate Professor

School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BCSE102P- Structured and object-oriented programming Laboratory

	Indicative Experiments
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-3: POINTERS**
  - **Declaration and Access of Pointer Variables**
  - **Pointer Arithmetic**
  - **Dynamic Memory Allocation**
  - **Pointers and Arrays**
  - **Pointers and Functions**

Pointers and arrays

- When 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.
- Base address is the location of first element (index 0) of the array.

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

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

Base Address

*int *p;// integer pointer pointing to array*
p=x; // Initialize with base address of x - equivalent to &x[0]
p+1= 1002
p+2=1004
p+3=1006 ……..

address of x[3] = base address + (3 x scale factor of int)
= 1000 + (3 x 2)
= 1006

Example- sum of elements – using pointers

```
void main()
{
        int *p, sum, i=0;
        int x[5]={1,2,3,4,5},
        p=x;  // integer pointer pointing to array
        printf("Element \t Value \t Address\n\n");
        while(i<5)
        {
                printf("x[%d] \t %d \t %u\n", i, *p, p);
                sum=sum+*p;
                i++;p++;
        }
        printf("\n Sum=%d\n", sum);
        printf("\n &x[0]=%u\n", &x[0]);
        printf("\n p=%u\n", p);
}
```

```
Element Value   Address

x[0]     1       6422000
x[1]     2       6422004
x[2]     3       6422008
x[3]     4       6422012
x[4]     5       6422016

 Sum=15

 &x[0]=6422000

 p=6422020
```

Example- Largest element– using pointers

```
void main()
{
        int i,j,n,a[25],*ptr;
        printf("\ Enter the number of elements : ");
        scanf("%d",&n);
        printf("\n Enter the array elements :  ");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        *ptr=a[0];
        for(i=0;i<n;i++)
        {
                if(a[i]>*ptr)
                        *ptr=a[i];
        }
    printf("\n The biggest element in the array is %d", *ptr);
}
```

```
Enter the number of elements :
5
Enter the array elements:
14
56
79
23
45
The biggest element in the array is 79
```

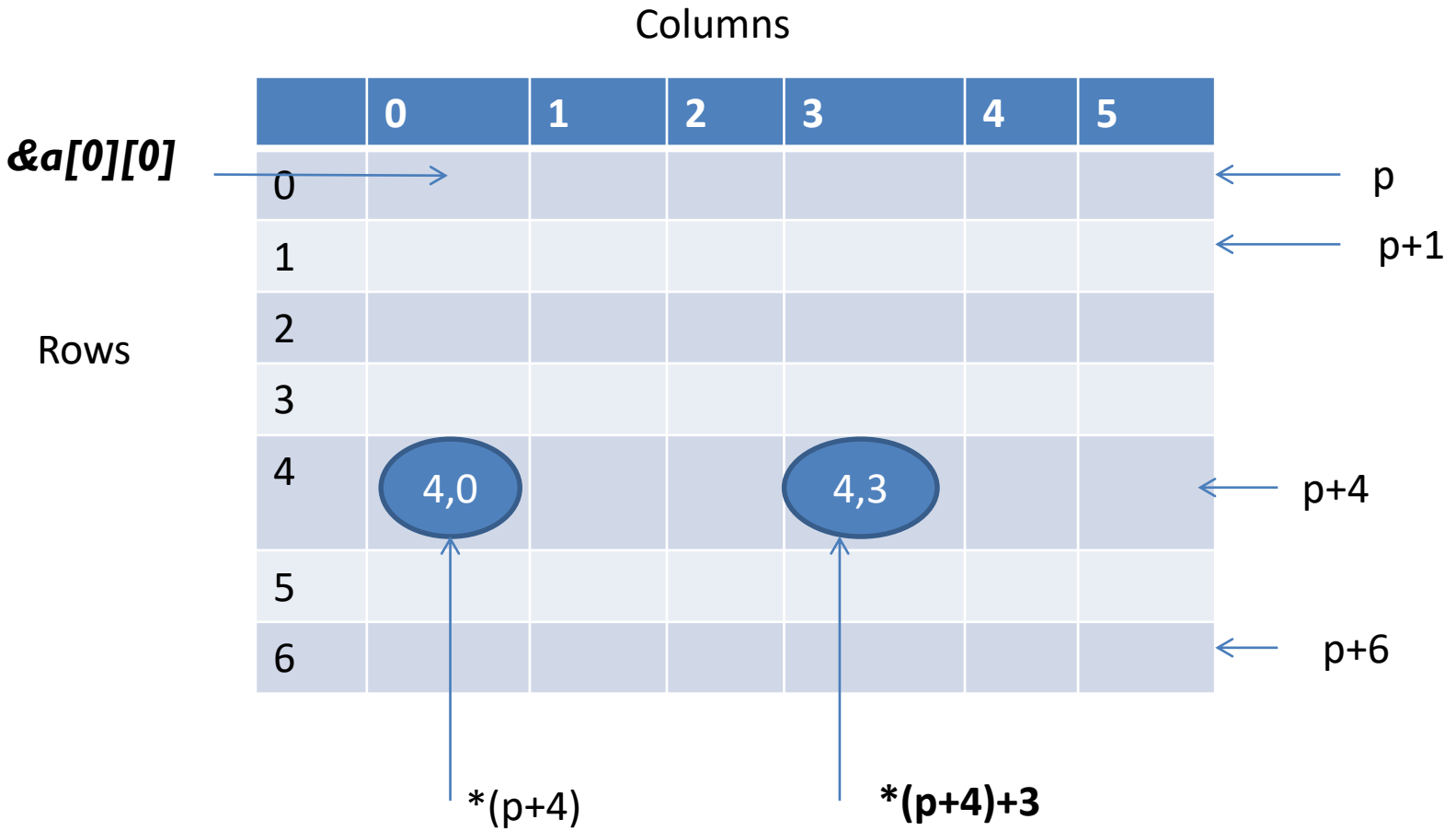
Representation in 2 D Array by pointers

p- pointer to first row

p+i – pointer to i^{th} row

**(p+i) – pointer to first element in the i^{th} row*

**(p+i)+j – pointer to jth element in the i^{th} row*



**(*(p+i)+j) – value stored in the cell(i,j)*

Also given by *(p+4x6)+3
= ***(p+27)**

Pointers and Character strings

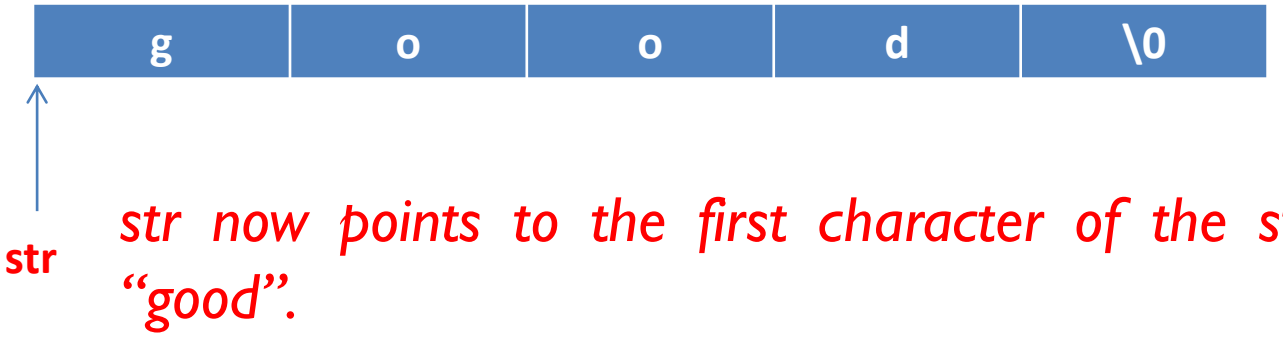
- Strings are treated like character arrays and therefore, they are declared and initialized as follows.

char str[5]= "good";

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

char *str= "good";

- Creates a string for the literal and then stores its address in the pointer variable **str.**



str now points to the first character of the string "good".

Array of Pointers to Character strings

- Consider an array of strings

  char city[3][20]= { }; ??

- **Allocates fixed storage of 60 bytes**
- **How memory is saved by using pointers??**

  char *city[3] = {"Vellore","Chennai","Bangalore"};

  (Array of 3 pointers to characters.)

It means

city[0] –Vellore

city[1] – Chennai

city[2] – Bangalore

- This declaration allocates only 27 bytes.

Example – Length of string using Pointers

```
int main()
{
        int length, i=0; char *chptr = “Welcome”;
        printf("%s\n",chptr);
        while(*chptr != '\0')
        {
                chptr++; i++;
        }
        printf("\n Length of the string is %d",i-1);
        return 0;
}
```

Pointers and functions- Passing Pointers

- Passing address of the variable as an argument to a function.**(Call by reference / call by address/ pass by pointers**)

```
main()
{
        int x=20;
        change(&x);  // Call by reference
        printf("%d\n",x);
}
change(int *p)
{
        *p=*p+10;
}
```

change() is called, address of the variable not its value

Inside function, p is declared as pointer and therefore p is the address of the variable x

Functions returning Pointers

- Functions can return single value by its name or multiple value through pointer parameters. **As like, functions can also return a pointer to the calling function**.

```
int *large(int *, int*);          // Function Declaration/Prototype
main()
{
        int a=10, b=20;
        int *p;
        p=large(&a,&b);   // Function Call
        printf("%d",*p);
}

int *large(int *x, int*y)                  // Function Implementation
{
        if(*x>*y)
                return (x);          // return address of a
        else
                return (y);          // return address of b
}
```

Pointer to a function

- Functions are stored in memory as like data.
- A function has a address, as like a variable.
- The function name represents the address of that function – so you can assign a pointer

- Declaration: Syntax

  datatype (*function_name) (datatype 1…..datatype n);

  Example : float *square ();

- Function call Prototype

  (*function_name) (datatype 1…..datatype n);

  If we remove the arguments it becomes functions returning pointer

  float *square();

Example – Pointer to a function

Normal Function Call

```
int sum(int,int);
int main()
{
printf("sum %d",sum(10,20));
return 0;
}
int sum(int x, int y)
{
int s= x+y;
return s;
}
```

Pointer to a function

```
int sum(int,int);
int main()
{
int (*p)(int,int);// Important
p=sum; or &sum
printf("sum %d",p(10,20));
return 0;
}
int sum(int x, int y)
{
int s= x+y;
return s;
}
```

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Thank You