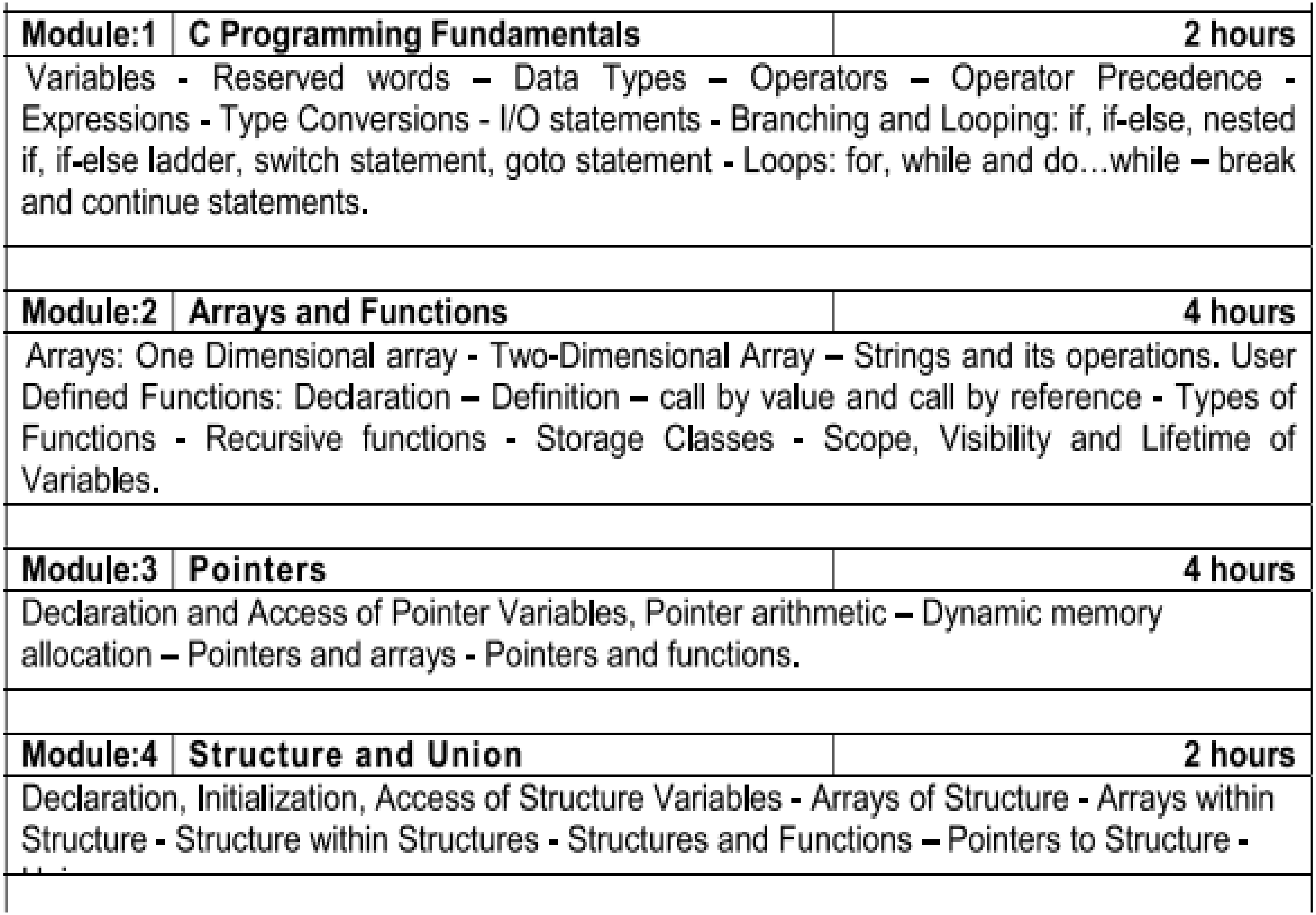
BCSE102L Structured and Object Oriented Programming

C PROGRAMMING LANGUAGE-MODULE-3

By,
Dr. S. Vinila Jinny,
ASP/SCOPE

SYLLABUS



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 -		

POINTER

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- Like any variable or constant, must declare a pointer before using it to store any variable address.

Syntax–

```
data type *var-name;
int *ip;
double *dp;
float *fp;
char *ch
```

ADVANTAGE OF POINTER IN C

- Pointer reduces the code and improves the performance.

- We can return multiple values from function using pointer.

- It make you able to access any memory location in the computer's memory.

SYMBOL USED IN POINTER

Symbol	Name	Description
& (ampersand sign)	address of operator	determines the address of a variable.
* (asterisk sign)	indirection operator	accesses the value at the address.

DECLARATION OF POINTER

Syntax:-

```
int *ptr;
int (*ptr)();
int (*ptr)[2];
```

For e.g.-

```
int a=5;             // a= variable name
int *ptr;           // value of variable= 5
ptr=&a;/* Address where it has stored in memory : 1025 (assume) */
```

A SIMPLE EXAMPLE OF C POINTER

```c
#include <stdio.h>
int main ()
{
int var = 20;
int *ip;
ip = &var;
printf("Address of var variable: %u\n", &var );
printf("Address stored in ip variable: %u\n", ip );
printf("Value of *ip variable: %d\n", *ip );
return 0;
}
```

OUTPUT

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

EXAMPLE

```
#include<stdio.h>
int main(){
int a=10,b=20,*p1=&a,*p2=&b;

printf("Before swap: *p1=%d *p2=%d",*p1,*p2);
*p1=*p1+*p2;
*p2=*p1-*p2;
*p1=*p1-*p2;
printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);

return 0;
}
```

NULL POINTER

- It is always a good practice to assign a NULL value to a pointer variable in case we do not have an exact address to be assigned.
- This is done at the time of variable declaration.
- A pointer that is assigned NULL is called a **null** pointer.
- The NULL pointer is a constant with a value of zero.

EXAMPLE

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;

   printf("The value of ptr is : %u\n", ptr  );

   return 0;
}
```

Output : The value of ptr is 0

POINTER ARITHMETIC

- Pointer holds address but can perform some arithmetic operations upon addresses.
- Not all arithmetic operations would be valid with them.
- There are only two arithmetic operations that we can use on pointers: addition and subtraction.

POINTER ARITHMETIC

- Let **p1** be an integer pointer with a current value of 2000. Also, assume **ints** are 4 bytes long.
- After the expression

```
p1++;
```

p1 contains 2004, not 2001.

- The reason for this is that each time **p1** is incremented, it will point to the next integer. The same is true of decrements.
- For example, assuming that **p1** has the value 2000, the expression

```
p1--;
```

causes **p1** to have the value 1996.

POINTER COMPARISON

- We can compare two pointers in a relational expression.
- For instance, given two pointers **p** and **q**,the following statement is perfectly valid:

```
if(p < q)
   printf("p points to lower memory than q\n");
```

- Generally, pointer comparisons are useful only when two pointers point to a common object, such as an array.

POINTERS AND ARRAYS

```
char str[80], *p1;
p1 = str;
```

- Here, **p1** has been set to the address of the first array element in **str**.
- To access the fifth element in **str**, we could write

```
str[4] or  *(p1+4)
```

- Main Difference is pointer arithmetic can be faster

POINTER ARITHMETIC ON ARRAYS

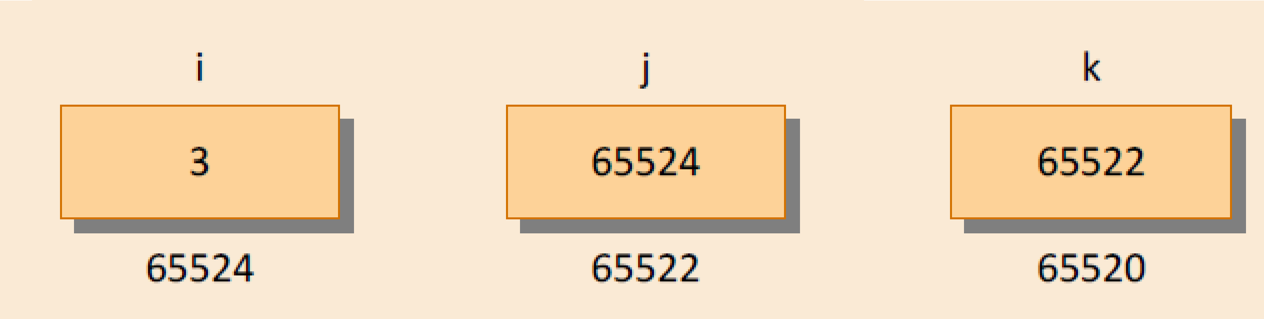
```
int main()
{
int N = 5;
int arr[] = { 1, 2, 3, 4, 5 };
int* ptr;
ptr = arr;
for (int i = 0; i < N; i++)
 {
printf("%d ", ptr[0]);          //printf("%d ", *ptr);
ptr++;
}}
```

POINTER ARITHMETIC

```
void traverseArr(int* arr, int N, int M)
{int i, j;
for (i = 0; i < N; i++) {
for (j = 0; j < M; j++) { printf("%d ", *((arr + i * M) + j));}
printf("\n");
}
}
int main()
{
int N = 3, M = 2;
int arr[][2] = { { 1, 2 },{ 3, 4 },{ 5, 6 } };
traverseArr(arr, N, M);
return 0;
}
```

POINTER TO POINTER

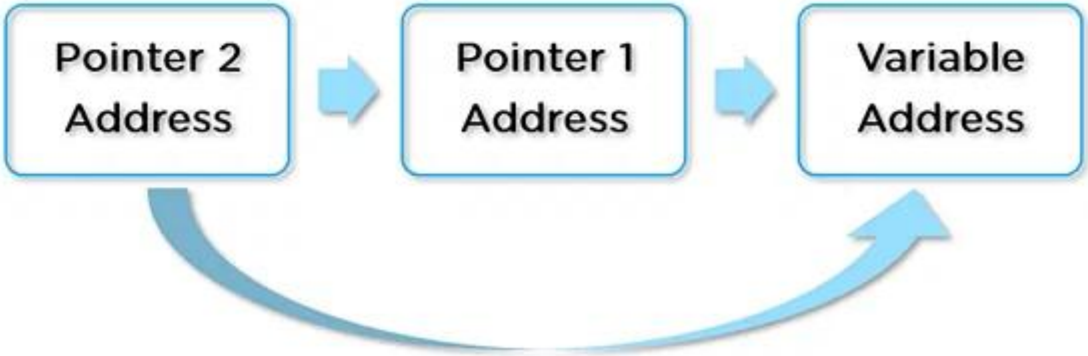
```
int  i = 3, *j, **k ;


j = &i ;
k = &j ;
```



i j k

3 65524 65522

65524 65522 65520



Pointer 2 Address

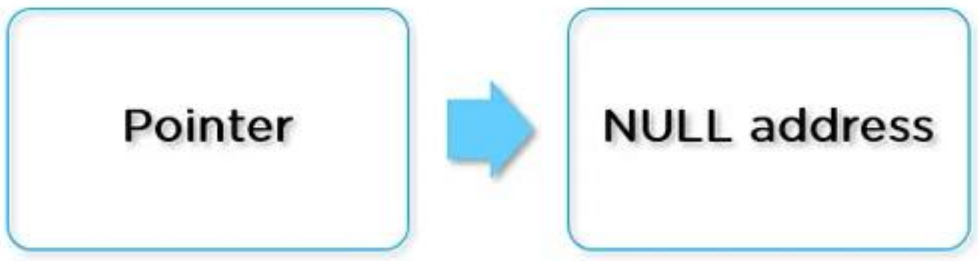
Pointer 1 Address

Variable Address

TYPES OF POINTER

- Null Pointer
- Void Pointer
- Wild Pointer
- Dangling Pointer

NULL POINTER

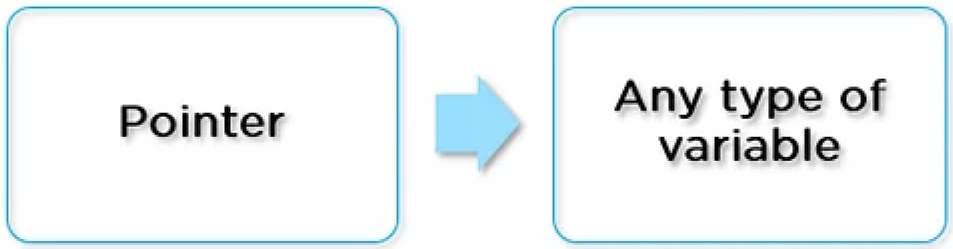


Pointer

NULL address

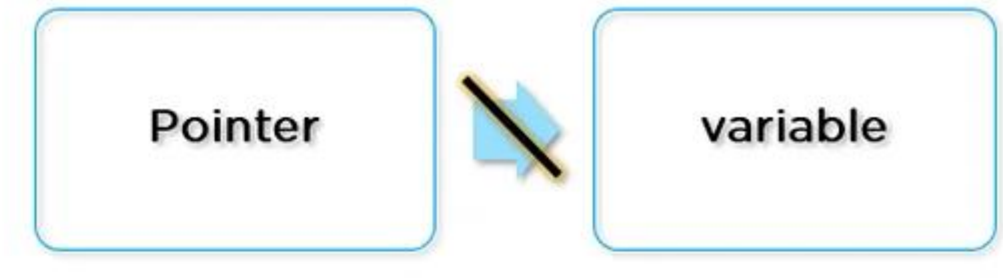
If you assign a NULL value to a pointer during its declaration, it is called Null Pointer.

VOID POINTER



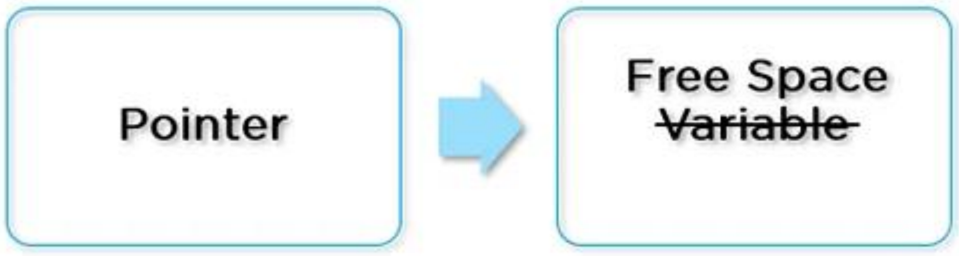
When a pointer is declared with a void keyword, then it is called a void pointer. To print the value of this pointer, you need to typecast it.

WILD POINTER



A wild pointer is only declared but not assigned an address of any variable. They are very tricky, and they may cause segmentation errors.

DANGLING POINTER



Suppose there is a pointer p pointing at a variable at memory 1004. If you deallocate this memory, then this p is called a dangling pointer.

INDEXING POINTERS

- An array name without an index is a pointer to the first element in the array.

```
char p[10];
```

- The following statements are identical:

```
p;
&p[0];
p == &p[0];
```

- evaluates to true because the address of the first element of an array is the same as the address of the array.

POINTER AS ARRAY

- An array name without an index generates a pointer.
- Conversely, a pointer can be indexed as if it were declared to be an array.

```
int *p, i[10];
p = i;
p[5] = 100; /* assign using index */
*(p+5) = 100; /* assign using pointer arithmetic */
```

POINTER AS ARRAY

- This same concept also applies to arrays of two or more dimensions.
- int a[10[10];
- a is equal to &a[0][0]
- a[0][4] is equal to *((int *)a+4).
- a[1][2] or *((int *)a+12).
- In general, for any two-dimensional array:
- a[j][k] is equivalent to *((base type *)a+(j*row length) + k

POINTER AS ARRAY

- A two-dimensional array can be reduced to a pointer to an array of one-dimensional arrays.
- Therefore, using a separate pointer variable is one easy way to use pointers to access elements within a row of a two-dimensional array.

POINTER AS ARRAY

```
int num[10] [10];
void pr_row(int j)
{
 int *p, t;
 p = (int *) &num[j] [0]; /* get address of first  element in row j */
 for(t=0; t<10; ++t) printf("%d ", *(p+t));
}
```

ARRAYS OF POINTERS

- Pointers can be arrayed like any other data type. The declaration for an int pointer array of size 10 is

```
int *x[10];
```

- To assign the address of an integer variable called var to the third element of the pointer array, write

```
x[2] = &var;
```

- To find the value of var, write

```
*x[2]
```

ARRAY OF POINTERS

- If we want to pass an array of pointers into a function, we can use the same method that we use to pass other arrays: Simply call the function with the array name without any subscripts.
- For example,
- a function that can receive array x looks like this:

```
void display_array(int *q[])
{
 int t;
 for(t=0; t<10; t++)
 printf("%d ", *q[t]);
}
```

HOW TO RETURN MULTIPLE VALUES FROM A FUNCTION IN C

- By using pointers.
- By using structures.
- By using Arrays.

RETURN MULTIPLE VALUES FROM A FUNCTION USING POINTER

```
#include <stdio.h>
void func(int *var1, int *var2, char *var3)
{// Function to return multiple values using pointers
    *var1 = 40;
    *var2 = 50;
    *var3 = 'X’;
}
int main(void)
{
    int var1, var2;
    char var3;
     func(&var1, &var2, &var3);
    printf("var1 = %d, var2 = %d, var3 = %c", var1, var2, var3);
    return 0;
}
```

HOW TO RETURN MULTIPLE VALUES FROM A FUNCTION USING ARRAY

```
#include <stdio.h>
// Function to return multiple values using an array
Void func(int *tempVar)
{
    *tempVar = 40;
    *(tempVar + 1) = 50;
    *(tempVar + 2) = 60;
}
int main(void)
{
    int var1, var2, var3;
    int arr[10];
    func(arr);
    var1 = arr[0];
    var2 = arr[1];
    var3 = arr[2];
    printf("var1 = %d, var2 = %d, var3 = %d", var1, var2, var3);
    return 0;
}
```

THANK YOU