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. |

# 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**


# Pointer

- Derived data type in C. Built from one of the fundamental data types in C.
- A variable which contains ***memory address*** of another variable as its value.
- Memory addresses are locations in the computer memory where program instructions and data are stored.
- Pointers can be used to access and manipulate the data stored in memory.
- A variable name references a value ***directly,*** whereas a pointer references a value ***indirectly.***
- Referencing a value through a pointer is called ***indirection.***
- A pointer variable must be declared before it can be used.


# Advantages of using Pointer

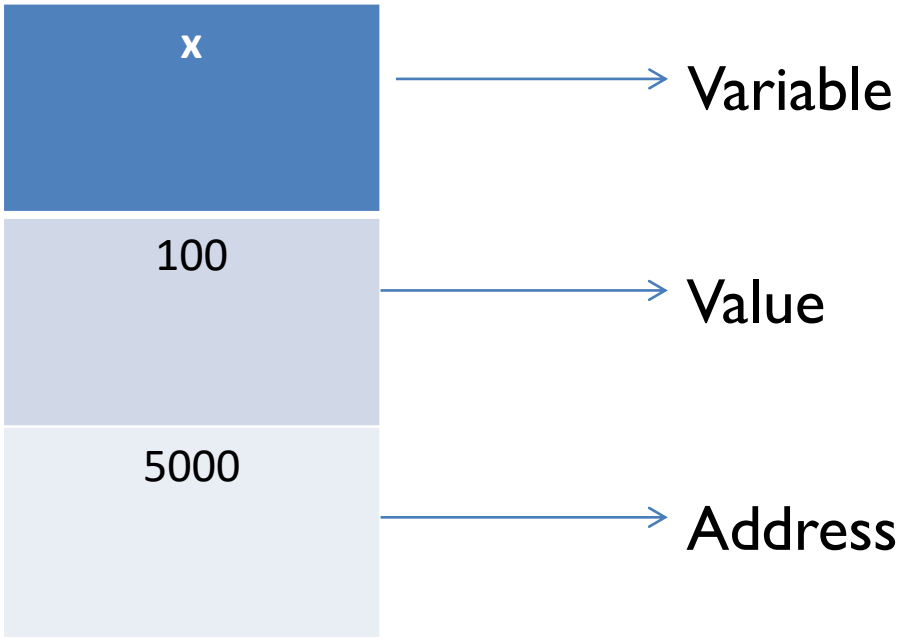
- It increases the execution speed and save data storage space.
- It is used to return multiple values from a function through function arguments.
- It supports Dynamic Memory Management.
- It is used to handle data structures like arrays, structure, linked lists, queues, stacks and trees.
- It reduces the length and complexity of programs.
- It improves the efficiency of program.
- Used to pass information back and forth between function and reference point.
- Provide easy ways to represent Multi Dimensional arrays.

# Representation of Variable

- When we declare a variable, the system allocates, somewhere in the memory, an appropriate location to hold the value of the variable.

- **For example :**

**int x= 100;**




# Pointer Variable

| variable | value | Address |
|---|---|---|
| x | 100 | 5000 |
| p | 5000 | 5048 |

- Since Memory addresses are simply numbers, they can be assigned to some variables, that can be stored in memory, like any other variable. Such variable that hold memory addresses are called pointer variables.


# Understanding Pointers

- **Declaring a Pointer Variable :**

  **Data-type *pt-name;**

- **pt-name** refers to valid pointer variable name. * is called as **indirection operator** also known as **dereferencing operator.**
- Data-type refers to a valid C data type. Pointer variable must be followed by '*' sign.
- **Example:**

```
int *x;   // integer pointer
float *f;  // float pointer
char *y;
```

- In the first statement **x is an pointer variable** and it tells the compiler that it holds the address of **any integer variable.**
- Normal variable provides direct access to its own values, whereas a **pointer provides indirect access to the values of the variable whose address it stores.**


# Example- Display address of a variable

```
#include<stdio.h>
void main()
{
        int  n;
        printf("enter the number:");
        scanf("%d", &n);
        printf(Value of n: %d\n", n);
        printf(Address of n:%u\n, &n);
}
```

***Reference Operator(&)- gives you the address of a variable***

**OUTPUT:**

| | | |
|---|---|---|
| **Enter the number** | **:** | **20** |
| **Value of N** | **:** | **20** |
| **Address of N** | **:** | **4066** |

# Initialization of Pointer Variables

- Initialization is the process of assigning the address of a variable to a pointer variable.
- Once a pointer variable has been declared we can use using '=' Assignment operator to initialize the variable.
- For example

```
int quantity;
int *p;                 /* Declaration*/
p=&quantity;            /*Initialization*/
```

***Dereference Operator (*)- gets you value from the address***

- Also Initialized as follows

```
int *p= &quantity;    /*Initialization*/
```

***Note: This is initialization for p , not for *p***


# Initialization of Pointer Variables

- Pointer variables always point to the corresponding type of data.

```
float a,b;
int x, *p;
p=&a;   // Wrong initialization.
```

*Prone to error- Assigning the address of a float variable to integer pointer.*

- Possible to combine the declaration of data variable, declaration of data variable and the initialization of pointer variable in one step.

```
int x, *p=&x;    // Valid initialization.
```

*Note: Remember that target variable x should be declared first.*

```
int  *p=&x, x;    //In-Valid initialization
```

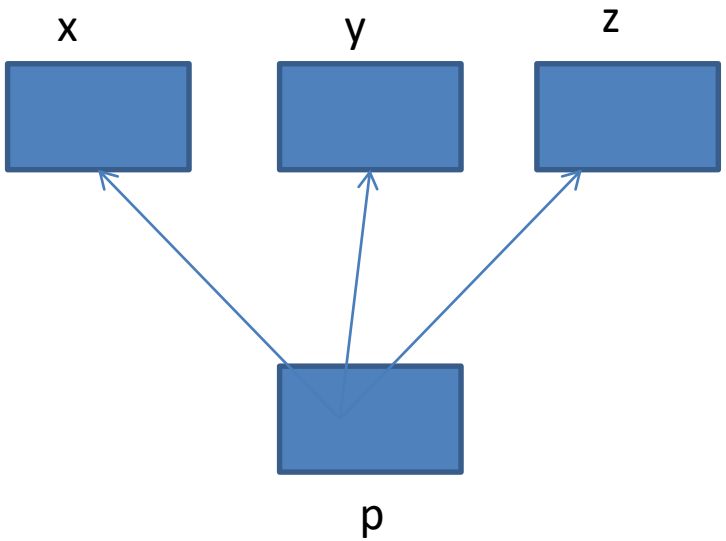
- Defining pointer variable with initial value of NULL or 0 is possible.

```
int *p= NULL;
int *p=0;
```

# Flexibility of Pointers

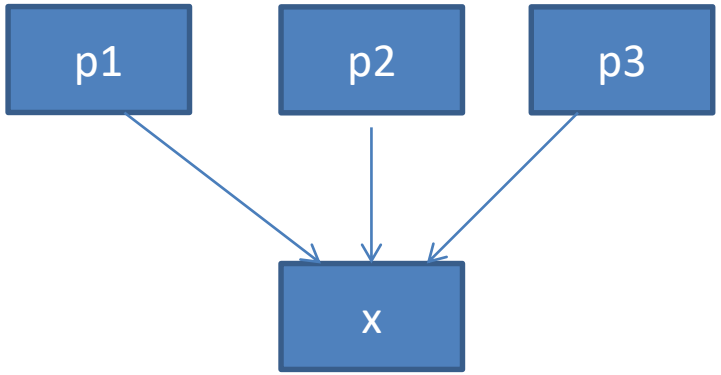
- Pointers are Flexible. We can make the same pointer to point to different data variables in different statements.

```
int x, y, z, *p;
……..
p=&x;
……..
p=&y;
……..
p=&z;
```




- Also different pointers to point to the same data variable.

```
int x;
int *p1=&x;
int *p2=&x;
int *p3=&x;
```




# Accessing variable through Pointers

- How to access the value of the variable using the pointer??
- Done by using * operator – **indirection operator/ dereferencing operator**

```
int mark, *p, n; // Declaration of integer and pointer variables
mark=560;  // assigns  560 to mark
p=&mark; // assigns  address of mark to pointer variable
n=*p;   //  pointer returns the value of the variable of
          which the pointer value is the address
          *p returns value of variable "mark" because
           p is the address of mark
```

**Note: * can be read as " value at address**


# Example

```
void main()
{
    int num, *intptr;
    float x, *floptr;
    char ch, *cptr;
    num=123;    x=12.34;    ch='a';
    intptr=&x;
    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);

}
```


# Example- How Pointer Works??

```
int main()
{
        int  *p, c;
        c = 10;
        printf("Address of c: %u\n", &c);
        printf("Value of c: %d\n\n", c);
        p = &c;
        printf("Address of pointer p: %u\n", p);
        printf("Content of pointer p: %d\n", *p);
        *p = 2;
        printf("Address of c: %u\n", &c);
        printf("Value of c: %d\n\n", c);
        return 0;
}
```

```
Address of c: 6422036
Value of c: 10
Address of pointer p: 6422036
Content of pointer p: 10
Address of c: 6422036
Value of c: 2
```


# Example- How Pointer Works when Function returning multiple values??

```
Void operation(int x, int y, int*s,int*d)
main()
{
        int  x=20, y=10,s,d;
        operation(x,y,&s,&d);
        printf("s=%d\n d=%d\n",s,d)
}

Void operation(int a, int b, int*sum,int*difference)
{
        *sum=a+b;
        *difference=a-b;
}
```


# Some Analysis in Pointers

int c, *p;

**p = c;**

// Wrong! ----- p is address whereas, c is not an address.

***p= &c;**

// Wrong! *p is the value pointed by address

**p = &c;**

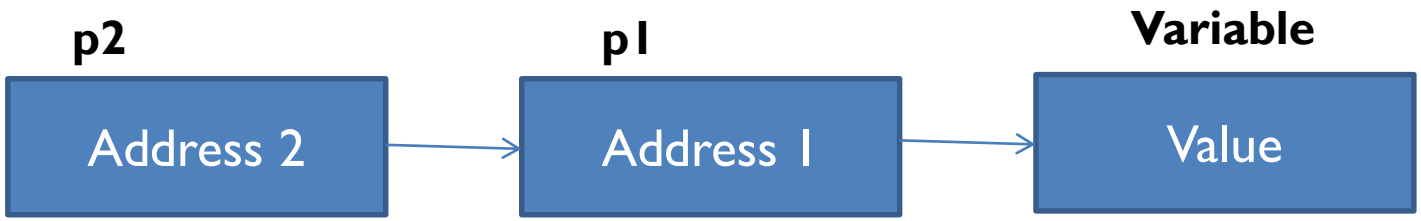
// Correct! p is an address and, &c is also an address.

***p = c;**

// Correct! *p is the value pointed by address and, c is also a value. .

# Chain of Pointers

- It is possible to make a **pointer to point to another pointer**.



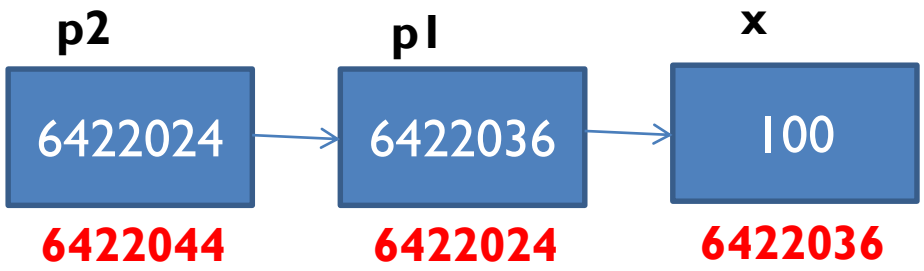

- **P2** contains address of pointer variable **p1** which points to the location that contains the desired value.
- Called as **multiple indirections.**
- A variable that is a **pointer to a pointer** must be declared using **additional indirection operator** symbols.

```
int **p2;
// p2 is a pointer to a pointer of int type
```

- p2 is not a pointer to an integer, it's a pointer to integer pointer.

# Example

```
main()
{
        int x, *p1, **p2;
        x = 100;
        p1 = &x;
        p2 = &p1;
        printf("%d %u",x,&x);
        printf("%d %u",p1,&p1);
        printf("%d",*p1);
        printf("%d%u",p2,&p2);
        printf("%d", **p2);
}
```




# Pointer Arithmetic


# Pointer Arithmetic

- Limited set of arithmetic operations can be performed on pointers.
- Pointer Arithmetic is slightly different from the ones that is generally used for mathematical calculations.
  1. Increment/Decrement of a Pointer(++,--)
  2. Addition of integer to a pointer(+ or +=)
  3. Subtraction of integer to a pointer(- or -=)
  4. Comparison of pointers of the same type.

- *Note: Pointer arithmetic is meaningless unless performed on an array. Adding two addresses makes no sense, because there is no idea what it would point to. Subtracting two addresses lets you compute the offset between these two addresses.*

# Increment/Decrement of a Pointer

```
const int MAX = 3;  // usage of const.
int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;     /* let us have array address in pointer */
   ptr = var;   //ptr = &var[0]
   for ( i = 0; i < MAX; i++)
   {
      printf("Address of var[%d] = %x\n", i, ptr );    //4000
      printf("Value of var[%d] = %d\n", i, *ptr );   //10
      ptr++;                          // ptr = ptr + 1 = 4004
   }
   return 0;
}
```

```
Address of var[0] = 61fe00
Value of var[0] = 10
Address of var[1] = 61fe04
Value of var[1] = 100
Address of var[2] = 61fe08
Value of var[2] = 200
```


```
int main()
{
        char string1[50],string2[50],*str1,*str2;
        int i, equal = 0;
        printf("Enter The First String: ");
        scanf("%s",string1);
        printf("Enter The Second String: ");
        scanf("%s",string2);
        str1 = string1;
        str2 = string2;
        while(*str1 == *str2)
                {
                        if ( *str1 == '\0' || *str2 == '\0' )
                                break;
                        str1++;
                        str2++;
                }
        if( *str1 == '\0' && *str2 == '\0' )
                printf("\n\nBoth Strings Are Equal.");
        else
                printf("\n\nBoth Strings Are Not Equal.");
}
```

***Example: Comparing Two strings using pointers***

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

# Thank You