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

BCSE102L- Structured and object-oriented programming

Module 1

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

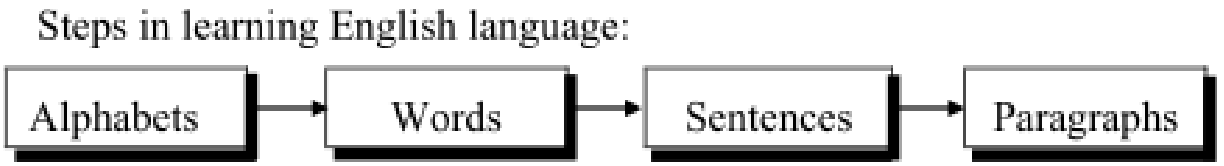
- **Module-1:C Program Fundamentals - Introduction**
  - **Variables**
  - **Reserved Words**
  - **Data types**
  - **Operators- Operator Precedence**
  - **Expressions**
  - **Type Conversions**
  - **I/O Statements**

BCSE102L- Structured and Object-Oriented Programming

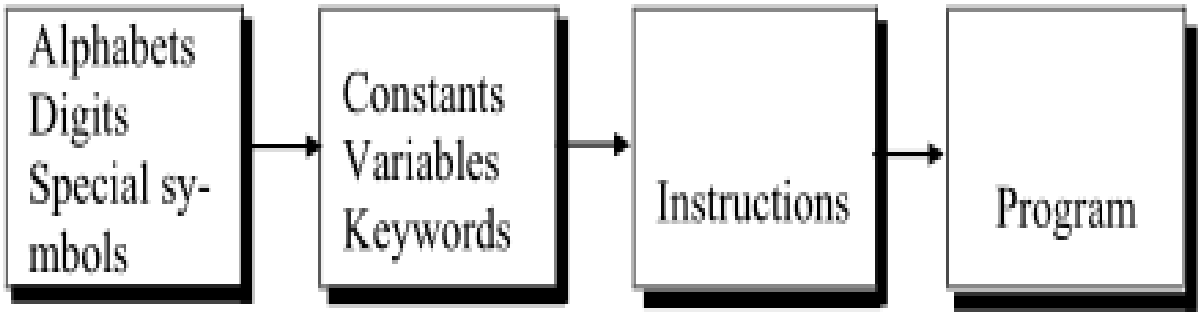
- **Module-1: C Program Fundamentals – Branching and Looping**
  - if, if-else, nested if, else-if ladder
  - Switch Statement
  - Goto Statement
  - Looping
    - For
    - While and do while
  - Break Statement
  - Continue Statement

Getting Started with C

- Communicating with a computer involves speaking the language the computer understands.
- What is the learning method of English??



- Steps in learning C



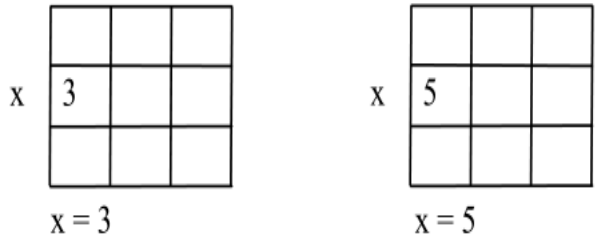
C Character Set

- A character denotes any alphabet, digit or special symbol used to represent information.

Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Constants, Variables and Keywords

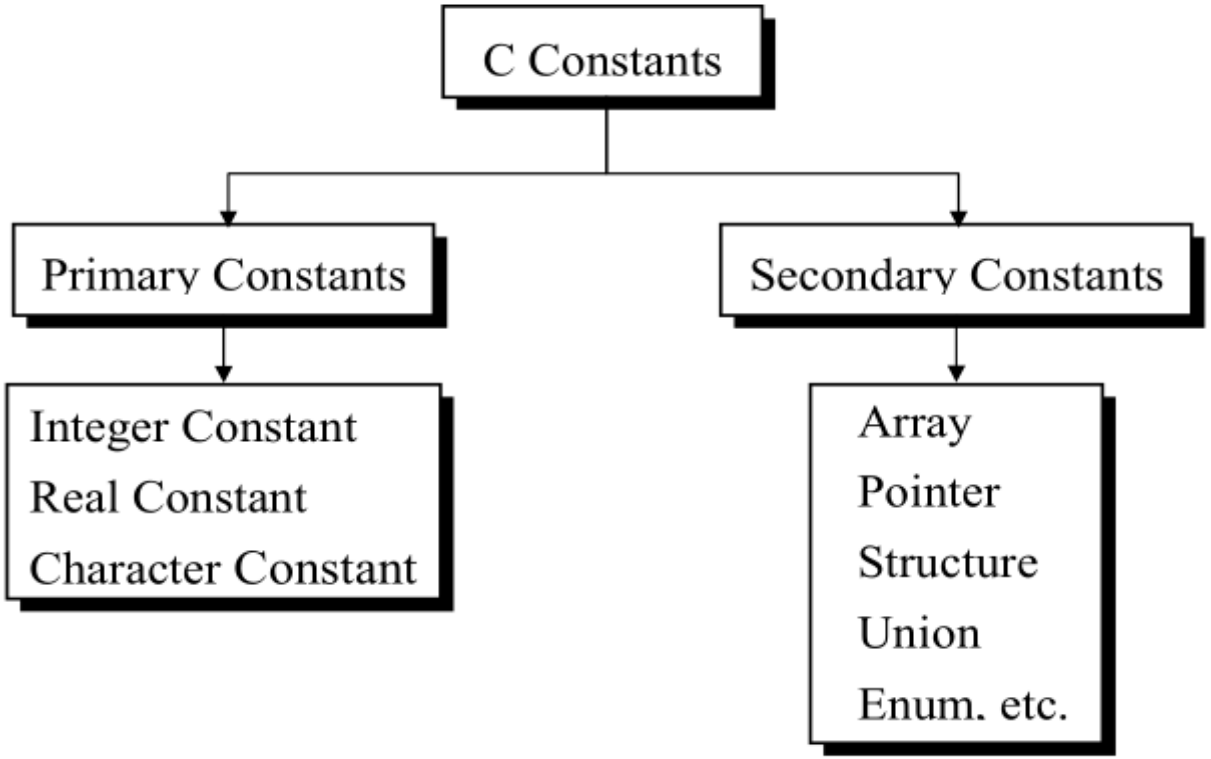
- Alphabets, Digits and special symbols when properly combined form constants, variables and keywords.
- Constant is an entity that doesn't change whereas a variable is an entity that may change.
- **For Example:** 3 is stored in a memory location and a name "x" is given to it , Then assign a new value 5 to the same memory location "x" – It overwrites earlier value 3 since memory location can hold only one value at a time.



- Location whose name is x can hold different values at different times so "x" is known as a variable.

Constants – Types

- **Constants can be divided into two major categories.**
  - Primary Constants
  - Secondary Constants



Rules for Constructing Integer Constants

1. An integer constant must have at least one digit.
2. It must not have a decimal point.
3. It can be either positive or negative.
4. If no sign precedes an integer constant, it is assumed to be positive.
5. No commas or blanks are allowed within an integer constant.
6. The allowable range for integer constants is -2147483648 to +2147483647

Note: Range of an Integer constant depends upon the compiler…

Example: 1234, +6789,-5000

Rules for Constructing Real Constants

- Real constants are often called Floating Point constants.
- Written in two forms—Fractional and Exponential form.

Fractional Form

1. A real constant must have at least one digit.
2. It must have a decimal point.
3. It could be either positive or negative.
4. Default sign is positive.
5. No commas or blanks are allowed within a real constant.

Example: +1234.56, 6789.0,-5000.89

Rules for Constructing Real Constants

Exponential Form

- Usually used if the value of the constant is either too small or too large
- Represented in two parts: Mantissa and Exponent
- For example 0.000123 can be written as 1.23e-4
- In Normal Arithmetic it is $1.23 * 10^{-4}$

- Rules:
  - Mantissa part and the exponential part should be separated by a letter **e or E.**
  - Mantissa part may have a positive or negative sign.
  - Default sign of mantissa part is positive.
  - Exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.
  - Range of real constants expressed in exponential form is-3.4e38 to 3.4e38.

Example: +3.2e-5, 4.1e8, -3.2e-5

Range specified in 32 bit GCC compiler

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld

Range specified in 32 bit GCC compiler

unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

Rules for Constructing Character Constants

1. A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
2. Both the inverted commas should point to the left.

   For example, 'A' is a valid character constant whereas 'A' is not.

Example: 'V', 'I' ,'T', '5', '$'

Variables- Types

- A **particular** type of variable can hold only the same type of constant.
  - An integer variable can hold only an integer constant.
  - Real variable can hold only a real constant.
  - Character variable can hold only a character constant.

Rules for Constructing Variable names

1. A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters.

Note : Avoid creating unnecessarily long variable

1. The first character in the variable name must be an **alphabet or underscore (_).**
2. No commas or blanks are allowed within a variable name.
3. No special symbol other than an underscore (**Example : net_pay**) can be used in a variable name.

Example: date_of_birth, basic_pay123, _book

Best Practice in using Variable names

Suppose you are writing a program for calculating simple interest, it is always advisable to construct meaningful variable names like "prin", "roi", "noy" to represent Principle, Rate of interest and Number of years rather than using the variables a, b, c.

Note: Hence rules for creating variable names remain same for all the types, How will you differentiate??

- Need of declaring the type of variable used

Example: int si, float roi, char gender etc

C KEYWORDS

- **Keywords** are the words whose meaning has already been explained to the C compiler.
- Keyword is a reserved word which cannot use it as variable name, constant name etc….
- 32 keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C KEYWORDS

- **Some Compilers Provide their own keywords**
- near, far, asm, etc
- Note : Every such compiler-specific keyword should be preceded by two underscores
- Example : __asm

Compiling and Linking

- **Preprocessing**. The program is first given to a preprocessor, which obeys commands that begin with # (known as **directives**).
  - **For Example : #include <stdio.h>, #include <stdlib.h>,**
- **Compiling**. The modified program now goes to a compiler, which translates it into machine instructions (**object code**).
- **Linking**. A linker combines the object code produced by the compiler with any additional code needed to yield a complete executable program.

test.c --->(compiled) --->**test.obj**(linked)--->**executable file**

Some Programming Terminologies

- **Source code:** The stuff you type into the computer/ The program you are writing.
- **Compile (build):** Taking source code and making a program that the computer can understand.
- **Executable:** compiled program that the computer can run.
- **Library:** Added functions for C programming which are bolted on to do certain tasks.
- **Header file:** Files ending in .h which are included at the start of source code.

Some Programming Terminologies

- **Compiler:**
  - Translates program written in “Source" language to "target" language/ Machine Understandable language
- **Interpreter:**
  - Translate and immediately execute
- **Assembler:**
  - Translate into machine language for particular machine

C- Program Example

```
/* Calculation of Simple Interest*/
# include <stdio.h>
int main( )
{
        int prin, noy ;
        float roi si ;
        prin = 1000 ;// Example for initialization of values
        noy = 3 ;
        roi = 8.5 ;
        /* formula for simple interest */
        si = prin * noy * roi / 100 ;
        printf ( "%f\n" , si ) ;
        return 0 ;
}
```

Try it yourself

1. To print your name and mobile number.
2. To print an integer by reading the value from the keyboard
3. To add two integers and display the sum.
4. To compute quotient and reminder when both the operands are integer. Repeat the same with both the operands as floating-point number.
5. To Print ASCII Value of a character
6. To Find the Size of int, float, double and char
7. Swapping of two numbers
8. To find area and perimeter of a rectangle.
9. If the marks obtained by a student in five different subjects are input through the keyboard, write a program to find out the total marks and percentage marks obtained by the student. Assume that the maximum marks that can be obtained by a student in each subject is 100.
10. To print even numbers from 1 to 100.

What is main()???

- It is a function – Container for set of statements.
- All statements that belong to main( ) are enclosed within a pair of braces { }.
- **Example:**

```
int main( )   //
{
    statement 1 ;
    statement 2 ;
    ……
    statement n ;
}
```

- Main function always returns an integer value

Purpose of printf()

- **printf()-** All output to screen is achieved using readymade library functions
- To use **printf() function** - Necessary to use **#include <stdio.h>** - **Called as preprocessor directives.**
- **SYNTAX:**

```
printf ( "<format string>", <list of variables> ) ;
```

%d – integer values
%f – float values
%c – character values
etc…….

Purpose of scanf()

- The Programs should ask the user to supply the values of any variable through the keyboard during execution is done by using **scanf** function.

Scanf("Control String", arg1, arg2, ...argn)

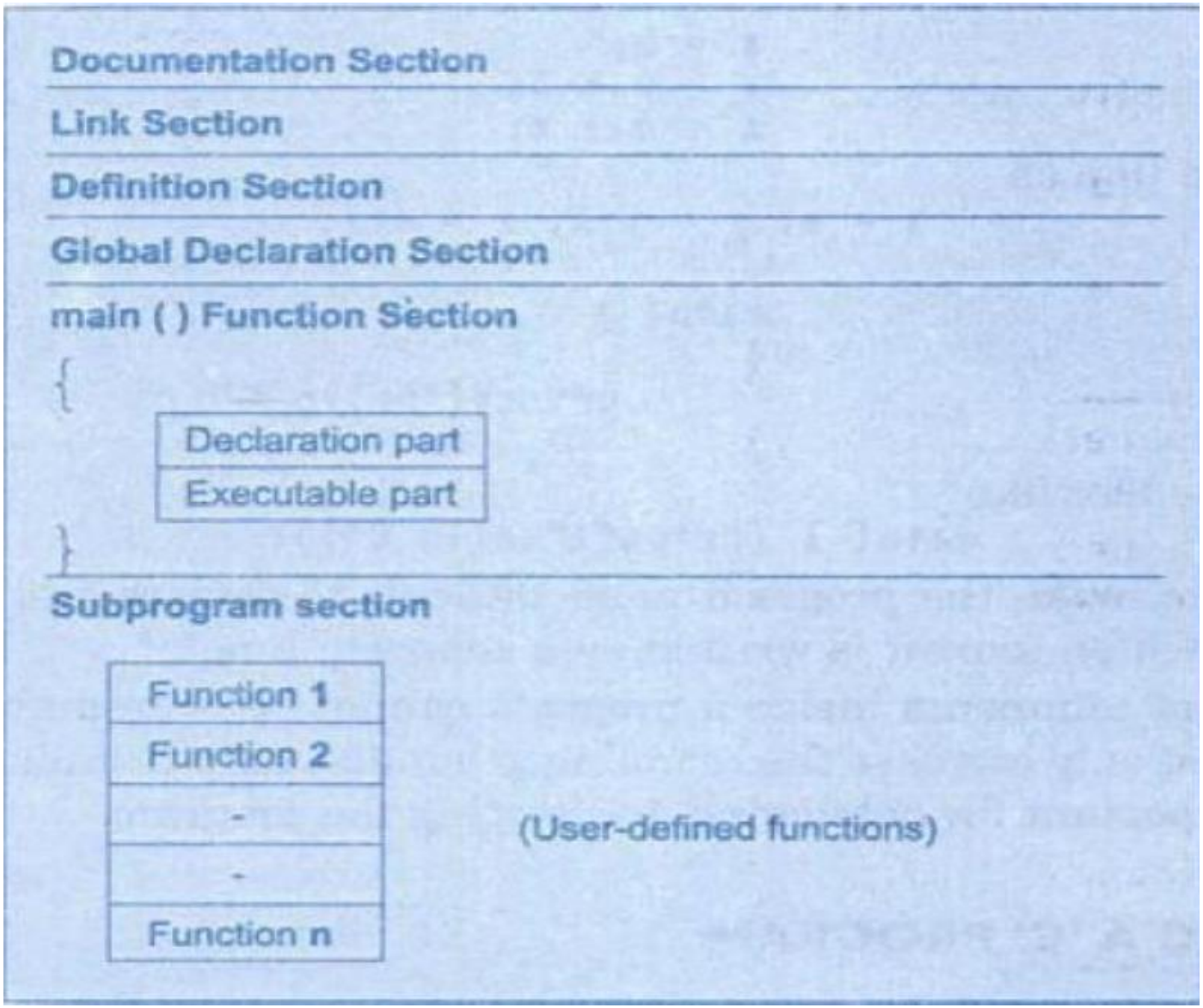
& - "Address of "- it gives location number(Address) Used by the variable in the memory

```
# include <stdio.h>
int main( )
        {
                int p, n ;
                float r, si ;
                printf ( "Enter values of p, n, r" ) ;
                scanf ( "%d %d %f", &p, &n, &r ) ;
                si = p * n * r / 100 ;
                printf ( "%f\n" , si ) ;
                return 0 ;
        }
```

INSTRUCTIONS

- **Types of Instructions**
  - **Type Declaration Instruction** – Instruction used to declare the type of variables used in a C program.
  - **Arithmetic Instruction** – Instruction used to perform arithmetic operations on constants and variables.
  - **Control Instruction** – Instruction used to control the sequence of execution of various statements in a C program.

STRUCTURE OF C PROGRAM



Managing Input and Output Operations

- Formatted Output
- Formatted Input

Formatted Output - printf()

- **SYNTAX:**

```
printf ( "<format string>", <list of variables> ) ;
```

Format string:

1. Characters that will printed on the screen
2. Format specifications that define the output format for display of each item
3. Escape Character sequences such as \n, \t, \b

List of Variables: Variables whose values are formatted and printed according to the specifications of the control string

```
% w.p type specifier
```

Integer number specifies total number of columns for the output value

Another integer number, that specifies the number of digits to the right of the decimal point

Output of Integer numbers

- **Example**

```
printf("%d\n",1234);    // Normal way
printf("%7d\n",1234); // right Justified
printf("%2d\n",1234);
printf("%-7d\n",1234); // left justified
printf("%07d\n",1234);// padding zeros
```

Note : For other data type, it may be printed by specifying in the place of format specifier.

Output of Real numbers

- **Examples**

```
float x= 12.3456;
printf("%f\n",x);
printf("%6.4 f\n",x);
printf("%6.2 f\n",x);
printf("%-6.2 f\n",x);
printf("%10.2e\n",x);
printf("%11.4e\n",-x);
printf("%-10.2e\n",x);
printf("%e\n",x);
```

% w.p f

% w.p e

Output of Characters/ Single character

```
char x= 'l';
printf("%c\n",x);
printf("%3c\n",x);
printf("%5c\n",x);
printf("%-3c\n",x);
printf("%-5c\n",x);
```

```
% wc
```

Output of Characters/ Strings

```
char name[25]="VIT  VELLORE  TAMILNADU"
printf("%s\n",name);
printf("%25s\n",name);
printf("%25.10s\n",name);
printf("%.5s\n",name);
printf("%-25.10s\n",name);
printf("%5s\n",name);
```

```
% w.ps
```

Mixed Data Output

```
printf("%d  %f %s %c", a,b,c,d);
```

Formatted Input - scanf()

- **SYNTAX:**

scanf(“Format string", <list of variables>) ;

Format string:

1. Data to be entered- consisting of % (conversion character), datatype character and optional number specifying the field width.

List of Variables/Arguments:

1. List of variable i.e arguments specify the address of the locations where the data to be stored.
2. Escape Character sequences such as \n, \t, \b

Inputting Integer numbers

% w d

- % - conversion character
- w- integer number that specifies field width of the number to be read.
- d – data type character
- **Example**

```
scanf("%d\n",&n);    // Normal way
scanf("%2d\n",&n);
scanf("%5d\n",&n);
scanf("%2d %5d\n",&n1,&n2); // 85  12345
scanf("%d %d\n",&n1,&n2);
```

Inputting Real numbers

- No specification for field width in real numbers.
- **Example**

```
scanf("%f %f%f\n",&n1,&n2,&n3);
```

Inputting Character strings

% ws Or
%wc

- **Example**

```
char name1[10], name2[20];
scanf("%15c\n",&name1);
scanf("%s\n",&name1);
scanf("%15s\n",&name2);
```

Reading Mixed Data types

- **Example:**

scanf("%s %c %d %f \n",&name,&gender,&age, &salary);

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

Thank You