Savitribai Phule Pune University

F. Y. B. C. A. (Science) Semester-II

BCA-126 Lab Course – II

Advance C WorkBook

Name: ____

CollegeName:_____________________________________

RollNo.:____________________ Division: ___________________________

AcademicYear:__________________________________

Editors:

Section-I:

1. **Patil P.U**
2. **Kulkarni P.P**
3. **Dani S.P**

Section-II:

1. **Jagdale D.V**
2. **More A.M**

Reviewed By:
Prof. Gangarde A.D

Introduction

1. **About the work book:**
   This workbook is intended to be used by F.Y.B.C.A. (Science) students for the C and RDBMS Assignments in Semester–II. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus.

2. **The objectives of this workbook are:**
   1) Defining the scope of the course.
   2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
   3) To have continuous assessment of the course and students.
   4) Providing ready reference for the students during practical implementation.
   5) Provide more options to students so that they can have good practice before facing the examination.
   6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. **How to use this workbook:**
   The workbook is divided into two sections. Section-I is related to Advance C assignments and Section-II is related to Introduction to Relational Database Management System.
   The Section-I (Advance C) is divided into Ten assignments. Each Advance C assignment has three SETs. It is mandatory for students to complete the SET A and SET B in given slot.
   The Section-II (Relational database Management System) is divided into fourteen assignments. Each assignment has three SETs. It is mandatory for students to complete SET A and SET B in given slot.

3.1 Instructions to the students

Please read the following instructions carefully and follow them.

1) Students are expected to carry this book every time they come to the lab for computer science practical.
2) Students should prepare oneself beforehand for the Assignment by reading the relevant material.
3) Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.

4) Studentswillbeassessedforeachexerciseonascalefrom0to5.

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

3.2 Instruction to the Instructors

1) Explain the assignment and related concepts in around ten minutes using whiteboard if required or by demonstrating the software.
2) You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
3) The value should also be entered on assignment completion page of the respective Lab course.

3.3 Instructions to the Lab Administrator

You have to ensure appropriate hardware and software is made available to each student.

The operating system and software requirements on server side and also client side areas given below:
1) Server and Client Side - (Operating System) Linux/Windows
2) Database server – PostgreSQL
3) Turbo C.

Table of Contents for Section-I

Assignment Completion Sheet

Lab Course I			
Advance C Assignments			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teachers Sign**
1	Use of Pre-Processor Directive		
2	Use Of Simple Pointers		
3	Advanced Use Of Pointers		
4	Concept Of Strings, Array Of Strings		
5	String Operations Using Pointers		
6	Command Line Arguments		
7	Structures(Using Array And Functions)		
8	Nested Structures And Unions		
9	Use Of Bitwise Operators		
10	File Handling		
Total (Out of 50)			
Total (Out of 15)			

CERTIFICATE

This is to certify that Mr./Ms.____

has successfully completed the Advanced programming in C laboratory

course in year __________ and his/her seat no is ____________. He/She

has scored _____ Marks out of 15.

Instructor	H.O.D. / Coordinator
Internal Examiner	External Examiner

Section I – Advanced Programming in C

Assignment No 1:- Study of Preprocessor Directive

1. File Inclusion Directive (#include)
2. Macro
2. # error and #pragma directives
3. Conditional Compilation
4. Predefined macros and Preprocessor operations.

Practice Programs

1) Example program for #define, #include preprocessors in C language:

```
#include <stdio.h>
#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define special_char '?'
 void main()
{
   printf("value  of  height  :  %d  \n",  height );
   printf("value of number : %f \n", number);
   printf("value of letter : %c \n", letter);
   printf("value of letter_sequence : %s \n", letter_sequence);
   printf("value of special_char: %c \n", special_char);
 }
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of special_char: ?
```

2) Example program for conditional compilation directives:

a) Example program for #ifdef, #else and #endifin C:

```
#include <stdio.h>
#define RAJU 100
 int main()
{      #ifdef RAJU
         printf("RAJU is defined. So, this line will be added in this C file\n");
```

```
  #else
  printf("RAJU is not defined\n");
  #endif
  return 0;  }
```

```
Output:   RAJU is defined. So, this line will be added in this C file
```

b) Example program for #ifndef and #endif in C:

```
#include <stdio.h>
#define RAJU 100
int main()
{
  #ifndef SELVA
  {
    printf("SELVA is not defined. So, now we are going to define here\n");
    #define SELVA 300
  }
  #else
    printf("SELVA is already defined in the program");

  #endif
  return 0;      }
```

Output:

```
SELVA is not defined. So, now we are going to define here
```

c) Example program for #if, #else and #endif in C:

```
#include <stdio.h>
#define a 100
int main()
{
  #if (a==100)
      printf("This line will be added in this C file since a = 100\n");
  #else
      printf("This line will be added in this C file since a is not equal to 100\n");
  #endif
  return 0;  }
```

Output:

```
This line will be added in this C file since a = 100
```

3) Example program for #pragma :

a) Illustration to suppress warning

```
    #include<stdio.h>
    #pragma warn –rvl
int main( )
    {
printf("It will not show any warning");
    }
```

Explanation :

- mean suppress warning message .

rvl means : "Function should return a value".

We have specified return type of main as “Integer” (By default it is integer)butwe are not going to return a value.
Usually this program will show warning message.
We have suppressed warning already (-rvl) so we won’t get any warning message.
4) Example program for #error :
The #error preprocessor directive indicates error. The compiler gives fatal error if #error directive is found and skips further compilation process.

```
#include<stdio.h>
       #ifndef MATH_H
       #error First include then compile
       #else
 void main( )
     {     float a;
           a=sqrt(7);
 printf("%f",a);
       }
       #endif
```

1) Predefined Macros

There are some predefined macros which are readily for use in C programming.

Predefined macro	Value
__DATE___	String containing the current date
__FILE___	String containing the file name
__LINE___	Integer representing the current line number
__STDC___	If follows ANSI standard C, then value is a nonzero integer
__TIME___	String containing the current date.

Example #1: predefined Macros

C Program to find the current time

```
#include <stdio.h>
int main()
{
   printf("Current time: %s",    TIME    );    //calculate the current time
}
```

Output: Current time: 19:54:39

2) The following printf statements display the values of the predefined macros **__LINE** , **FILE** , **TIME** , and **DATE__**and print a message indicating the program's conformance to ANSI/ISO standards based on **STDC** :

```
/*     This example illustrates some predefined macros.*/

#pragma langlvl(ANSI)
#include <stdio.h>
#if   STDC___
#   define CONFORM     "conforms"
#else
#   define CONFORM     "does not conform"
#endif
int main(void)
{
  printf("Line %d of file %s has been executed\n", LINE , FILE );
  printf("This file was compiled at %s on %s\n", TIME , DATE );
  printf("This program %s to ANSI/ISO standard C\n", CONFORM);
}
```

3) Stringize (#):
The stringize or number-sign operator ('#'), when used within a macro definition, converts a macro parameter into a string constant. This operator may be used only in a macro that has a specified argument or parameter list.
When the stringize operator immediately precedes the name of one of the macro parameters, the parameter passed to the macro is enclosed within quotation marks and is treated as a string literal.

Example 1:
```
#include <stdio.h>
#define   message_for(a, b)             printf(#a " and " #b ": How areyou!\n")
int main(void)
{
   message_for(Carole, Debra);
   return 0;
}
```

Output: Carole and Debra: How are you!

Example 2:
```
#include<stdio.h>
#define string(s) #s
int main(){
    char str[15]= string(World is our );
    printf("%s",str);
    return 0;
}
```

Output: World is our

Explanation : Its intermediate file will look like:

```
int main(){
    char str[15]=”World is our”;
    printf("%s",str);
    return 0;
}
```

Argument of string macro function ‘World is our’ is converted into string by the operator # .Now the string constant “World is our” is replaced the macro call function in line number 4.

4) Token Pasting (##):

The token-pasting operator (##) within a macro definition combines two arguments. It permits two separate tokens in the macro definition to be joined into a single token.
If the name of a macro parameter used in the macro definition is immediately preceded or followed by the token-pasting operator, the macro parameter and the token-pasting operator are replaced by the value of the passed parameter. Text that is adjacent to the token-pasting operator that is not the name of a macro parameter is not affected.

Example1 :

```
#define tokenpaster(n) printf ("token" #n " = %d", token##n)
tokenpaster(34);
```

This example results in the following actual output from the preprocessor:

```
printf ("token34 = %d", token34);
```

This example shows the concatenation of token##n into token34. Both the stringize and the token-pasting operators are used in this example.

Example 2 :

```
#include<stdio.h>
#define merge(p,q,r) p##q##r
int main(){
    int merge(a,b,c)=45;
    printf("%d",abc);
    return 0;
}
```

Output : 45

Explanation:

Arguments a,b,c in merge macro call function is merged in abc by ## operator .So in the intermediate file declaration statement is converted as :

```
int abc = 45;
```

5) Multiline macros

By using a the "\" to indicate a line continuation, we can write our macros across multiple lines tomake them a bit more readable.

For instance, we could rewrite swap as

```
#define SWAP(a, b)  {                    \
                   a ^= b;         \
                   b ^= a;         \
                   a ^= b;         \
               }
```

Notice that you do not need a slash at the end of the last line! The slash tells the preprocessor that the macro continues to the next line. Aside from readability, writing multi-line macros may make it more obvious that you need to use curly braces to surround the body because it's more clear that multiple effects are happening at once.

6) Defined

The special operator defined is used in '#if' and '#elif' expressions to test whether a certain name is defined as a macro. defined *name* and defined (*name*) are both expressions whose value is 1 if *name* is defined as a macro at the current point in the program, and 0 otherwise. Thus, #if defined MACRO is precisely equivalent to #ifdef MACRO.

defined is useful when you wish to test more than one macro for existence at once. For example,

```
#if defined (var) || defined (ns16000)
```

would succeed if either of the names var or ns16000 is defined as a macro.

Conditionals written like this:

```
#if defined BUFSIZE && BUFSIZE >= 1024
```

can generally be simplified to just #if BUFSIZE >= 1024, since if BUFSIZE is not defined, it will be interpreted as having the value zero. If the defined operator appears as a result of a macro expansion, the C standard says the behavior is undefined

Set A

1) Write the Program to implement macros for example:-define constant and array size

2) Write the Program to

   a. find maximum of two integers
   b. check whether a number is positive ,negative or Zero
   d. check given number is even or odd

3) Write the Program to illustrate the use of following using #pragma

Sr.No	Warning Message	Code
1	Code has no effect	eff
2	Function should return a value	rvl
3	Parameter ‘parameter’ is never used	par
4	Possible use of ‘identifier’ before definition	def
5	Possibly incorrect assi gnment	pia
6	Unreachable code	rch
7	Non portable pointer conversion	rpt

Set B

1) Write the Program to

   a. find minimum of three numbers using nested macros
   b. swap two numbers

Set C

1) Write the Program to find cube of a number using nested macros

Assignment Evaluation

0: Not Done [ ]　　1: Incomplete [ ]　　2: Late Complete [ ]

3: Needs Improvement [ ]　　4: Complete [ ]　　5: WellDone [ ]

Signature of Teacher

Assignment 2:- Use of Pointer Variables

Pointer :

A pointer provides a way of accessing a variable without referring to the variable directly.

The address of the variable is used.

Syntax:-

```
data_type *var_name;
```

Example : int *p; char *p;

Where, * is used to denote that "p" is pointer variable and not a normal variable

This definition set aside two bytes in which to store the address of an integer variable and gives this storage space the name p.

Key points to remember about pointers in C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- The value of null pointer is 0.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).

Valid Pointer Arithmetic Operations

- Adding a number to pointer.
- Subtracting a number form a pointer.
- Incrementing a pointer.
- Decrementing a pointer.
- Subtracting two pointers.
- Comparison on two pointers.

Invalid Pointer Arithmetic Operations

- Addition of two pointers.
- Division of two pointers.
- Multiplication of two pointers.

Example:

```
#include<stdio.h>
#include<conio.h>

void main() {
    int var = 10, *ptr;
    char c_var = 'A', *c_ptr;
  float f_var = 4.65, *f_ptr;

    /* Initialize pointers */
    ptr = &var;
    c_ptr = &c_var;
    f_ptr = &f_var;

    printf("Address of var = %u\n", ptr);
    printf("Address of c_var = %u\n", c_ptr);
    printf("Address of f_var = %u\n\n", f_ptr);

    /* Incrementing pointers */
    ptr++;
    c_ptr++;
    f_ptr++;
    printf("After increment address in ptr = %u\n", ptr);
    printf("After increment address in c_ptr = %u\n", c_ptr);
    printf("After increment address in f_ptr = %u\n\n", f_ptr);

    /* Adding 2 to pointers */
    ptr = ptr + 2;
    c_ptr = c_ptr + 2;
    f_ptr = f_ptr + 2;

    printf("After addition address in ptr = %u\n", ptr);
    printf("After addition address in c_ptr = %u\n", c_ptr);
    printf("After addition address in f_ptr = %u\n\n", f_ptr);

    getch();
    return 0;
}
```

Sample output:

Assume the addresses to be-

```
Address of var = 2293300
Address of c_var = 2293299
Address of f_var = 2293292
```

```
After incrementing address in ptr = 2293304
After incrementing address in c_ptr = 2293300
After incrementing address in f_ptr = 2293296

After addition address in ptr = 2293312
After addition address in c_ptr = 2293302
After addition address in f_ptr = 2293304
```

Set A

1) Write a program to Interchange values of two numbers using pointers
2) Write a program to display the elements of an array containing n integers in reverse order using pointer
3) Write a program to reverse the elements of an array containing n integers using pointer
4) Write a program to calculate average marks of "n" number of students using pointers and array

.

Set B

1) Write a program to accept an array and a number. Check whether the number is present in the array ,print the index number of the first occurrence of the number.
2) In above program --If number is present calculate the number of occurrences of that number in the array using pointers.
3) Write a program to accept an array sort the given array. (Using pointer)
4) Write a program to accept a matrix of size 3x3 and print the same using pointer.

Set C

1) Write a program to accept a matrix of size 3x3 and print transpose of the matrix using pointer.
2) Write a program to accept two matrices of size 3x3 and print the addition using pointer.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Teacher

Assignment 3:- Advanced use of Pointers

C Array of Pointer

Just like array of integers or characters, there can be array of pointers too.
An array of pointers can be declared as :

```
<type> *<name>[<number-of-elements];
```

For example :

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

Constant Pointer

A constant pointer is declared as :

```
<type-of-pointer> *const <name-of-pointer>
```

For example :

```
#include<stdio.h>

int main(void)
{
   char ch = 'c';
   char c = 'a';
   char *const ptr = &ch;// A constant pointer
   ptr = &c;// Trying to assign new address to a constant pointer. WRONG!!!!
   return 0;
}
```

Gives a compilation error

Pointer to a constant

Syntax :

```
const <type-of-pointer> *<name-of-pointer>;
```

For example :

```
#include<stdio.h>

int main(void)
{
   char ch = 'c';
const char *ptr = &ch; // A constant pointer 'ptr' pointing to 'ch'
   *ptr = 'a';// WRONG!!! Cannot change the value at address pointed by 'ptr'.
   return 0;
```

}
Compilation error

Multiple Indirection :

C permits the pointer to point to another pointer. This creates many layers of pointer and therefore called as multiple indirection. A pointer to a pointer has declaration similar to that of a normal pointer but have more asterisk(*) before them. This indicates the depth of the pointer

For Example: Double Pointer

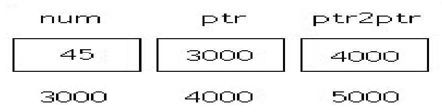
Declaration :

```
int **ptr2ptr;
```

Example:

```
int num = 45,*ptr,**ptr2ptr;
Ptr=&num;
Ptr2ptr=&ptr;
```

Diagrammatic Representation:



Statement	What will be the Output ?
*ptr	45
**ptr2ptr	45
ptr	&num
ptr2ptr	&ptr

Passing pointer to function and Returning pointer from function

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function, hence if you return a pointer connected to a local variable, that pointer be will pointing to nothing when function ends.

Example :-

```
#include <stdio.h>
#include <conio.h>
int* larger(int*, int*);
void main()
{
 int a=15;
 int b=92;
 int *p;
 p=larger(&a, &b);
 printf("%d is larger",*p);
}
int* larger(int *x, int *y)
{
 if(*x > *y)
  return x;
 else
  return y;}
```

Pointer to functions

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

type (***pointer-name**)(*parameter*);

Example :

```
int (*sum)();    //legal declaraction of pointer to function
int *sum();    //This is not a declaraction of pointer to function
```

Example:-

```
#include <stdio.h>
#include <conio.h>

int sum(int x, int y)
{
 return x+y;
}

int main( )
{
 int (*fp)(int, int);
 fp = sum;
 int s = fp(10, 15);
 printf("Sum is %d",s);
 getch();
 return 0;
}
```

Output : 25

Function pointer

void *(*foo) (int*) ;

It appears complex but it is very simple. In this case **(*foo)** is a pointer to the function, whose return type is **void*** and argument is of **int*** type.

Example Program:

```
/*program to add two numbers using function pointer */
#include<stdio.h>

int sum (int n1,int n2);
int main()
{
int num1=10;
int  num2=20;
int result;

int*(*ptr)(int,int);
ptr=&sum;
result=(*ptr)(num1,num2);
printf(Addition is : %d",result);
return 0;
}

int sum(int n1,int n2)
{
    return(n1+n2)
}
```

Output :Addition:20

Dynamic Memory Allocation:

The functions used are:

malloc()- Allocates requested size of bytes and returns a pointer pointing to first byte of allocated space
Syntax:-
ptr= (cast type *)malloc(Element_size);

calloc()-Allocates multiple blocks of memory each of same size of bytes and returns a pointer pointing to first byte of allocated space .Sets all bytes to zero.
Syntax:-
ptr= (cast type *)calloc(n,Element_size);

realloc()- reallocate previously allocated memory
syntax:-
 ptr= (cast type *)realloc(pointer,new_Element_size);

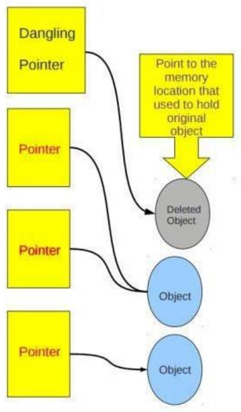
free()- to free the previously allocated memory
syntax:-
 free(pointer);

```
/*Program to create memory for int , char and float variables at run time*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
         int *ivar;
        char *cvar;
        float *fvar;

// Allocating memory dynamically
ivar=(int *)malloc(1*sizeof(int));
cvar=(char *)malloc(1*sizeof(char));
fvar=(float *)malloc(1*sizeof(float));

printf("Enter the integer value :");
scanf("%d",&ivar);

printf("Enter the character value :");
scanf("%c",&cvar);

printf("Enter the float value :");
scanf("%f",&fvar);

printf("The inputted value are %d %c %f :",ivar,cvar,fvar);


/* free the allocated memory*/
free(ivar);
free(cvar);
free(fvar);

getch();
}
```

What is Dangling Pointer?

While programming, we use pointers that contain memory addresses of data objects. Dangling pointer is a pointer that points to the memory location even after its de-allocation. Or we can say that it is pointer that does not point to a valid data object of the appropriate type. The memory location pointed by dangling pointer is known as dangling reference.

Now if we access the data stored at that memory location using the dangling pointer then it will result in program crash or an unpredictable behavior.



Let's take an example to understand this.

Cause of Dangling Pointer in C

```
void function(){
   int *ptr = (int *)malloc(SIZE);
   . . . . . .
   . . . . . .
   free(ptr);    //ptr now becomes dangling pointer which is pointing to dangling reference
}
```

In above example we first allocated a memory and stored its address in ptr. After executing few statements we de-allocated the memory. Now still ptr is pointing to same memory address so it becomes dangling pointer.

How to Solve Dangling Pointer Problem in C?

To solve this problem just assign NULL to the pointer after the de-allocation of memory that it was pointing. It means now pointer is not pointing to any memory address.

```
void function(){
   int *ptr = (int *)malloc(SIZE);
   . . . . . .
   . . . . . .
   free(ptr);    //ptr now becomes dangling pointer which is pointing to dangling
reference
   ptr=NULL;    //now ptr is not dangling pointer
}
```

What is Memory leak in C and how can we avoid it.

Memory leak happens when programmer allocated memory in heap but don't release it back to the heap. Memory leak reduces the available memory for program and as a result the performance of program reduces.

Here is an example of *memory leak*:

```
#include <stdio .h>
#include <stdlib.h>
Void getSometing(){
  /* Dynamically declare memory for an integer array of 10 elements */
  int*array = (int*) malloc(10*sizeof(int));
  /* Do something and return without releasing allocated memory */
  return;
}

int main() {
   int i;
   for(i = 0; i<100; i++){
      getSometing();
   }
```

```
    return0;
}
```

In above program, function getSometing( ) dynamically allocates memory an array but then returns without de-allocating it. Every time getSometing( ) function is called it reduces the available memory for the program. To avoid memory leaks, getSometing( ) function should release allocated memory using free.

```
voidgetSometing(){
   /* Dynamically declare memory for an integer array of 10 elements */
   int*array = (int*) malloc(10*sizeof(int));
   /* Do something and release allocated memory */
   free(array);
   return;
}
```

SET A

1) Write a program to multiply two numbers using function pointer
2) Write a Program to accept an array and print the same using double pointer
3) Write a program to calculate average of array of n numbers .Pass the array to a function and use pointers
4) Write a program to accept an array and a number .Write a function to find the number of occurrences of number in the array.(using pointer)

SET B

1) Write a program to read 1 D array of "n" elements and print the inputted array element (using dynamic memory allocation)
2) Write a program to find sum of n elements entered by user. To perform this, allocate memory dynamically using malloc() function
3) Write a program to find sum and average of n elements entered by user. To perform this, allocate memory dynamically using calloc() function.
4) Write a program to find largest among "n" numbers using dynamic memory allocation.

SET C

1) Write a program to accept a 2D array and print the addition of all elements (allocate memory at run time)

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Teacher

Assignment 4:- Concept of Strings , Array of Strings

String is a sequence of characters that is treated as a single data item and terminated by null character '\0'. Remember that C language does not support strings as a data type. A **string** is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

Declaring and Initializing a string variables

```
char name[13] = "C Prgramming";        // valid character arrayinitialization

char name[10] = {'L','e','s','s','o','n','s','\0'};      // valid initialization
```

C supports a format specification known as the **edit set conversion code %[..]** that can be used to read a line containing a variety of characters, including white spaces.

```
#include<stdio.h>
#include<string.h>
void main()
{   char str[20];
    printf("Enter a string");
    scanf("%[^\n]", &str); //scanning the whole string, including the white spaces
    printf("%s", str);
}
```

Another method to read character string with white spaces from terminal is by using the gets() function.

```
char text[20];
gets(text);
printf("%s", text);
```

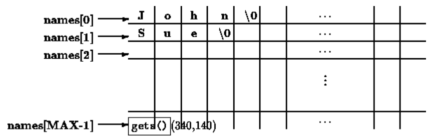
Arrays of String

Besides data base applications, another common application of two dimensional arrays is to store an array of strings. In this section we see how an array of strings can be declared and operations such as reading, printing and sorting can be performed on them.

A string is an array of characters; so, an array of strings is an array of arrays of characters. Of course, the maximum size is the same for all the strings stored in a two dimensional array. We can declare a two dimensional character array of MAX strings of size SIZE as follows:

char names[MAX][SIZE];

Since names is an array of character arrays, names[i] is the character array, i.e. it points to the character array or string, and may be used as a string of maximum size SIZE - 1. As usual with strings, a NULL character must terminate each character string in the array. We can think of an array of strings as a table of strings, where each row of the table is a string as shown in Figure below



Set A

1) Write a program to find the length of a string.
2) Write a program to copy a string into another.
3) Write a program to reverse a string by passing it to a function.
4) Write a program to check whether a given string is palindrome or not. (Write a function which accepts a string and returns 1 if it is palindrome and 0 otherwise)
5) Write a program to concatenate two strings.

Set B

1) Write a program to find the number of vowels ,consonants, digits and white space in a string.
2) Write a program that accepts names of n cities and write functions for the following:
   a) Search for a city
   b) Display the longest names
3) Write a program which accepts a sentence from the user and replaces all lower case letters by uppercase letters.
4) Write a program to find the First Capital Letter in a String .write a function iscap() to find the first capital letter.
5) Write a program to remove all other characters in a string except alphabets.

Set C

1) Write a program to accept a word and a string .Remove / delete the given word from a string.

Example:- if word is= "Hello " and the String is "Hello All Well Come"

The output is:- "All Well Come"

2) Write a program to print the words ending with letter d in the given sentence(multiword string).

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Teacher

Assignment No 5:- String operations using Pointers

To demonstrate string operations using pointers

C – string.h library functions

All C inbuilt functions which are declared in string.h header file are given below. The source code for string.h header file is also given below for your reference.

List of inbuilt C functions in string.h file:

String functions	**Description**
strcat ( )	Concatenates str2 at the end of str1
strncat ( )	Appends a portion of string to another
strcpy ( )	Copies str2 into str1
strncpy ( )	Copies given number of characters of one string to another
strlen ( )	Gives the length of str1
strcmp ( )	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2
strcmpi ( )	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
strchr ( )	Returns pointer to first occurrence of char in str1
strrchr ( )	last occurrence of given character in a string is found
strstr ( )	Returns pointer to first occurrence of str2 in str1
strrstr ( )	Returns pointer to last occurrence of str2 in str1
strdup ( )	Duplicates the string
strlwr ( )	Converts string to lowercase
strupr ( )	Converts string to uppercase
strrev ( )	Reverses the given string
strset ( )	Sets all character in a string to given character
strnset ( )	It sets the portion of characters in a string to given character
strtok ( )	Tokenizing given string using delimiter

Example program for strcat( ) function in C:

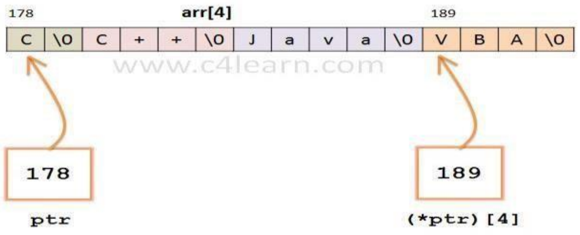
In this program, two strings "fresh2refresh" and "C tutorial" are concatenated using strcat( ) function and result is displayed as "C tutorial fresh2refresh".

```
#include<stdio.h>
#include<string.h>
int main( )
{
        char source[]="For Student";
        char target[]=”C Tutorial”;

        printf(“\n Source String =%s”,source);
        printf(“\n Target String =%s”,target);
        strcat(target,source);
         printf(“\n Target SAtring after strcat()=%s”,target);
}
```

```
Sample Output:
Source String                          =For Student
Target String                    = C Tutorial
Target String after strcat( )=C Tutorial For Student
```

Pointer to array of strings

A pointer which is pointing to an array whose content are strings,is known as pointer to array of strings.



With reference to above example

1. ptr   : It is pointer to array of string of size 4.
2. array[4] : It is an array and its content are string.

```
/*program to print the Address of the Character Array */

#include<stdio.h>
int main()
{
int i;
char * arr[4]={“C”,”C++”,”JAVA”,”VBA”};
char *(*ptr)[4]=&arr[0];
for(i=0;i<4;i++)
    printf(“Address of String %d : %u\n”,i+1,(*ptr)[i]);
retutn 0;
```

```
Output:
Address of string1=178
Address of String2=180
Address of String3=184
Address of String4=189
```

Set A

1) Write a program to Calculate Length of the String using Pointer.

2) Write a program to print Contents of 2-D character array(using pointer to array of string)Refer to above example.

3) Write a function similar to strlen that can handle unterminated strings.
Hint: you will need to know and pass in the length of the string.

4) Write a program to compare only left most ‘n’ characters from the given string .Accept n and string to be compared from the user.(using pointer)

5) Write a program to
   -compare two strings using library function
   -sets the portion of characters in a string to given character using library function

Set B

1) Write a program to compare two strings. If they are not equal display their length and if equal concatenate them

2) Write a program to pass two strings to user defined function and copy one string to another using pointer

3) Write a program to reverse string, without using another string variable.

4) Write a program to accept a string and a substring and check if the substring is present in the given string

Set C

1) Write a program to Count number of words in the given sentence .

2) Write a program to concatenate only the leftmost n characters from source with the destination string.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Teacher

Assignment No 6:-Command Line Arguments

Command line arguments in C:

main() function of a C program accepts arguments from command line or from other shell scripts by following commands. They are,

- argc
- argv[]

where,

argc – Number of arguments in the command line including program name
argv[] – This is carrying all the arguments

Example

```
#include<stdio.h>
#include<conio.h>
     void main(int argc,char* argv[])
     { clrscr();
       printf("\n Program name : %s \n", argv[0]);
       printf("1st arg : %s \n", argv[1]);
       printf("2nd arg : %s \n", argv[2]);
       printf("3rd arg  : %s \n", argv[3]);
       printf("4th arg : %s \n", argv[4]);
       printf("5th arg : %s \n", argv[5]);
     getch();}
```

Set A

1) Write a program to display the arguments passed using command line argument(refer to above example).
2) Write a program to add two numbers using Command Line Arguments.

Set B

1) Write a program to calculate the factorial of one number by using the command line argument.
2) Write a program to Generate Fibonacci Series of N Numbers using Command-Line Argument.

Set C

1) Write a program to accept three integers as command line arguments and find the minimum, maximum and average of the three numbers. Display error message if the number of arguments entered are invalid.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Teacher

Assignment No 7:-Structures (Using Arrays and Functions)

Structure

C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

- If you want to access structure members in C, structure variable should be declared.
- Many structure variables can be declared for same structure and memory will be allocated for each separately.
- It is a best practice to initialize a structure to null while declaring, if we don’t assign any values to structure members.

Array of Structures

- C Structure is collection of different datatypes (variables) which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

Passing structure to a function

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won’t be available to other functions unless it is passed to those functions by value or by address(reference).
- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

Passing structure to function in C:

It can be done in 3 ways.

1. Passing structure to a function by value
2. Passing structure to a function by address(reference)
3. No need to pass a structure – Declare structure variable as global

Structure using Pointer

C structure can be accessed in 2 ways in a C program. They are,

1. Using normal structure variable
2. Using pointer variable

Dot(.) operator is used to access the data using normal structure variable and arrow (->) is used to access the data using pointer variable.

<u>Set A</u>

1) Write a program to store and access "id, name and percentage" for one student.

2) Write a program to create student structure having fields roll_no, stud_name, mark1, mark2, mark3.
Calculate the total and average of marks

3) Write a program to create student employee having field emp_id, emp_name, designation.
Pass this entire structure to function and display the structure elements.

4) Write a program to copy one structure to another structure .(using pointer)

<u>Set B</u>

1) Write a program to store and access "id, name and percentage" for 3 students.(array of structures)

2) Write a program to create a student structure having fields roll_no, stud_name and address.
Accept the details of 'n' students into the structure, rearrange the data in alphabetical order of
stud_name and display the result.

3)Create an employee structure(eno, ename, salary). Write a menu driven program to perform the following operations (using function)

a. Add employee
b. Display all employees having salary>10000

<u>Set C</u>

1) Write a menu driven program in 'C' that shows the working of a library.
The menu option should be

- Add book information.
- Display book information.
- List all books of given author.
- List the count of books in the library.
- Exit.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Teacher

Assignment No 8 :-Nested Structures and Unions

Nested Structure

- Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.
- The structure variables can be a normal structure variable or a pointer variable to access the data. You can learn below concepts in this section.

1. Structure within structure in C using normal variable
2. Structure within structure in C using pointer variable.

Union

Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

- Union and structure in C are same in concepts, except allocating memory for their members.
- Structure allocates storage space for all its members separately.
- Whereas, Union allocates one common storage space for all its members
- We can access only one member of union at a time. We can’t access all member values at the same time in union. But, structure can access all member values at the same time. This is because,Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.
- Many union variables can be created in a program and memory will be allocated for each union variable separately.

pointer to union

pointer is special kind of variable which is capable of storing the address of a variable in c programming. Pointer which stores address of union is called as pointer to union.

Set A

1) Write a program to declare a structure "employee"(name,age,salary) which contains another structure "address"(house number, street) as member variable .Accept the details of one employee and display it.(using normal variable)

2) Write a program to declare a structure "employee"(name,age,salary) which contains another structure "address"(house number,street) as member variable.Accept the details of one employee and display it.(using pointer variable)

3) Write a program to to store and access “ name, subject and percentage” for two student.(using union)

4) Write a program to store and access “ name, subject and percentage” for one student.(using pointer to union)

Set B

1) Write a program to create a union for a library book with the following details (id , title , publisher , cost).If the code is 1 store the number of copies,if code= 2 store the issue month name and if code=3 store the edition number.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Teacher

Assignment No 9 :-Use of Bitwise Operators

```
* C Program to demonstrate use of bitwise operators */
#include<stdio.h>
int main()
{
    unsigned char a = 5, b = 9; // a = 5(00000101), b = 9(00001001)
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a&b); // The result is 00000001
    printf("a|b = %d\n", a|b); // The result is 00001101
    printf("a^b = %d\n", a^b); // The result is 00001100
    printf("~a = %d\n", a = ~a);    // The result is 11111010
    printf("b<<1 = %d\n", b<<1); // The result is 00010010
    printf("b>>1 = %d\n", b>>1); // The result is 00000100
    return 0;
}
```

Output:

```
a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b1 = 4
```

Following are interesting facts about bitwise operators.

1) The left shift and right shift operators should not be used for negative numbers
The result of is undefined behabiour if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined. Also, if the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits

The bitwise XOR operator is the most useful operator from technical interview perspective. It is used in many problems. A simple example could be "Given a set of numbers where all elements occur even number of times except one number, find the odd occurring number" This problem can be efficiently solved by just doing XOR of all numbers.

```
// Function to return the only odd occurring element
int findOdd(int arr[], int n) {
   int res = 0, i;
   for (i = 0; i < n; i++)
     res ^= arr[i];
   return res;
}
```

```
int main(void) {
   int arr[] = {12, 12, 14, 90, 14, 14, 14};
   int n = sizeof(arr)/sizeof(arr[0]);
   printf ("The odd occurring element is %d ", findOdd(arr, n));
   return 0;
}
// Output: The odd occurring element is 90
```

Set A

1) Write a program to swap two numbers using bitwise operators

2) write a menu driven program in 'C' to accept a number from user and perform following operations on it

   i) Right Shift
   ii) Left Shift
   iii) One's complement

Set B

1) Given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in list. One of the integers is missing in the list. Write a program to find the missing integer.

Example:

I/P [1, 2, 4, ,6, 3, 7, 8]
O/P 5

2) Write a program to check whether the given Number is Palindrome or not using Bitwise Operator

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Teacher

Assignment No 10 :-File Handling

File Handling(Text Files)

You should read the following topics before starting this exercise

1. Concept of streams
2. Declaring a file pointer
3. Opening and closing files
4. File opening modes
5. Random access to files

Operations performed	**Syntax**	**Example**
Declaring File pointer	FILE * pointer;	FILE *fp;
Opening a File	fopen(“filename”,mode); where mode = “r”, “w”, “a”, “r+”, “w+”, “a+”	fp=fopen(“a.txt”, “r”);
Checking for successful open	if (pointer==NULL)	if(fp==NULL) exit(0);

Checking for end of file
 	
Closing a File
	feof
	
	fclose(pointer); fcloseall();
		if(feof(fp)) printf(“File has ended”);
		fclose(fp);

Character I/O	fgetc, fscanf
	fputc, fprintf
		ch=fgetc(fp);
		fscanf(fp, ”%c”,&ch);
		fputc(fp,ch);

String I/O	fgets, fscanf
	fputs, fprintf
		fgets(fp,str,80);
		fscanf(fp, ”%s”,str);

Reading and writing formatted data	fscanf
	fprintf
		fscanf(fp, ”%d%s”,&num,str);
		fprintf(fp, “%d\t%s\n”, num, str);

Random access to files	ftell, fseek, rewind	fseek(fp,0,SEEK_END); /* end of file*/ long int size = ftell(fp);

File Handling (Binary File)

In binary files, information is written in the form of binary . All data is written and read with no interpretation and separation i.e. there are no special characters to mark end of line and end of file.
I/O operations on binary files

Reading from a binary file	fread(address,size-of-element,number of elements,pointer);	fread (&num,sizeof(int),1,fp); fread (&emp,sizeof(emp),1,fp); fread(arr,sizeof(int),10,fp) ;
Writing to a binary file	fwrite(address,size-of-element,number of elements,pointer);	fwrite (&num,sizeof(int),1,fp); fwrite (&emp,sizeof(emp),1,fp);

Set A

1) Write a program to create a file, read its contents and display on screen with each case of character reversed.

2) Write a program to create a file called emp.rec and store information about a person in terms of his name, age and salary.

3) Write a program to accept two filenames as command line arguments. Copy the contents of the first file to the second such that the case of all alphabets is reversed.

4) Write a program to write data of 5 employees to a binary file and then read the file.

5) Write a program to count the number of lines in a file.

Set B

1) Write a program to delete specific line from a file

2) Write a program to Compare two Binary Files, Printing the First Byte Position where they Differ

3) Write a menu driven program to create a structure student (roll number, name, percentage) . Perform the following operations on a binary file- “student.dat”.
- Add a student (Note: Students should be assigned roll numbers consecutively)
- Search Student
    - according to roll number
- Display all students

<u>Set C</u>

1) Write a menu driven program for a simple text editor to perform the following operations on a file, which contains lines of text.

i. Display the file
ii. Copy m lines from position n to p
iii. Delete m lines from position p
iv. Modify the nth line
v. Add n lines

2) Create two binary files such that they contain roll numbers, names and percentages. The percentages are in ascending orders. Merge these two into the third file such that the third file still remains sorted on percentage. Accept the three filenames as command line arguments

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Teacher