**KPIT TECHNOLOGIES**

**WEEKLY REPORT**

**WEEK 3- Report (DATE: 07/6/2024)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student name** | **Week** | **Branch** | **USN** |
| **Sunayan Dutta** | **3** | **CS (CSE)** | **1NH20CS220** |

**Yashavant Kanetkar Book**

**Question 101-150:**

**101. Write a program to reverse the strings stored in an array of pointers to strings**.

ANS:

#include <stdio.h>

#include <string.h>

void reverseString(char\* str) {

int len = strlen(str);

int i, j;

char temp;

for (i = 0, j = len - 1; i < j; i++, j--) {

temp = str[i];

str[i] = str[j];

str[j] = temp;

}

}

int main() {

char\* arr[] = {"Hello", "World", "Programming", "in", "C"};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original strings:\n");

for (int i = 0; i < n; i++) {

printf("%s\n", arr[i]);

}

for (int i = 0; i < n; i++) {

reverseString(arr[i]);

}

printf("\nReversed strings:\n");

for (int i = 0; i < n; i++) {

printf("%s\n", arr[i]);

}

return 0;

}

**102. A stack is a data structure in which addition of new element or deletion of existing element**

**always takes place at the same end known as ‘top’ of stack. Write a program to implement**

**a stack using a linked list.**

ANS:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void push(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = head;

head = newNode;

}

int pop() {

if (head == NULL) {

printf("Stack is empty\n");

return -1;

}

int data = head->data;

struct Node\* temp = head;

head = head->next;

free(temp);

return data;

}

int peek() {

if (head == NULL) {

printf("Stack is empty\n");

return -1;

}

return head->data;

}

int isEmpty() {

return head == NULL;

}

int main() {

push(10);

push(20);

push(30);

printf("%d popped\n", pop());

printf("Top element is %d\n", peek());

return 0;

}

**103. In a data structure called queue the addition of new element takes place at the end (called ‘rear’ of queue), whereas deletion takes place at the other end (called ‘front’ of queue).**

**Write a program to implement a queue using a linked list.**

ANS:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* front = NULL;

struct Node\* rear = NULL;

void enqueue(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

if (isEmpty()) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

int dequeue() {

if (isEmpty()) {

printf("Queue is empty\n");

return -1;

}

struct Node\* temp = front;

int data = temp->data;

front = front->next;

if (front == NULL) {

rear = NULL;

}

free(temp);

return data;

}

int isEmpty() {

return front == NULL;

}

int main() {

enqueue(10);

enqueue(20);

enqueue(30);

printf("%d dequeued\n", dequeue());

printf("Front element is %d\n", front->data);

return 0;

}

**104. Write a program to read a file and display its contents along with line numbers before each line.**

ANS:

#include <stdio.h>

int main(int argc, char\* argv[]) {

if (argc < 2) {

printf("Usage: %s <filename>\n", argv[0]);

return 1;

}

FILE\* fp = fopen(argv[1], "r");

if (fp == NULL) {

printf("Error opening file: %s\n", argv[1]);

return 1;

}

int line\_number = 1;

char line[256];

while (fgets(line, sizeof(line), fp) != NULL) {

printf("%d. %s", line\_number, line);

line\_number++;

}

fclose(fp);

return 0;

}

**105. Write a program to append the contents of one file at the end of another.**

ANS:

#include <stdio.h>

int main(int argc, char\* argv[]) {

if (argc < 3) {

printf("Usage: %s <source\_file> <destination\_file>\n", argv[0]);

return 1;

}

FILE\* source\_fp = fopen(argv[1], "r");

if (source\_fp == NULL) {

printf("Error opening source file: %s\n", argv[1]);

return 1;

}

FILE\* dest\_fp = fopen(argv[2], "a");

if (dest\_fp == NULL) {

printf("Error opening destination file: %s\n", argv[2]);

fclose(source\_fp);

return 1;

}

char ch;

while ((ch = fgetc(source\_fp)) != EOF) {

fputc(ch, dest\_fp);

}

printf("Contents of %s appended to %s successfully.\n", argv[1], argv[2]);

fclose(source\_fp);

fclose(dest\_fp);

return 0;

}

**106. The information about colors is to be stored in bits of an unsigned char variable called color. Bit numbers 0 to 6, each represent 7 colors of a rainbow, i.e., bit 0 represents violet, 1 represents indigo, and so on. Write a program that asks the user to enter a number and based on this number it reports which colors in the rainbow do the number represents**

ANS:

# include <stdio.h>

void display ( unsigned short int time ) ;

int main( )

{

unsigned short int time ;

puts ( "Enter any number less than 24446: " ) ;

scanf ( "%hu", &time ) ;

display ( time ) ;

return 0 ;

}

void display ( unsigned short int tm )

{

unsigned short int hours, minutes, seconds, temp ;

hours = tm >> 11 ;

temp = tm << 5 ;

minutes = temp >> 10 ;

temp = tm << 11 ;

seconds = ( temp >> 11 ) \* 2 ;

printf ( "For Time = %hu\n", tm ) ;

printf ( "Hours = %hu\n", hours ) ;

printf ( "Minutes = %hu\n", minutes ) ;

printf ( "Seconds = %hu\n", seconds ) ;

}

**107. The time field in a structure is 2 bytes long. Distribution of different bits which account for hours, minutes and seconds is given in Figure 21.6.Define a function that would receive the 2-byte time and print the equivalent hours, minutes and seconds.**

ANS:

# include <stdio.h>

void display ( unsigned short int time ) ;

int main( )

{

unsigned short int time ;

puts ( "Enter any number less than 24446: " ) ;

scanf ( "%hu", &time ) ;

display ( time ) ;

return 0 ;

}

void display ( unsigned short int tm )

{

unsigned short int hours, minutes, seconds, temp ;

hours = tm >> 11 ;

temp = tm << 5 ;

minutes = temp >> 10 ;

temp = tm << 11 ;

seconds = ( temp >> 11 ) \* 2 ;

printf ( "For Time = %hu\n", tm ) ;

printf ( "Hours = %hu\n", hours ) ;

printf ( "Minutes = %hu\n", minutes ) ;

printf ( "Seconds = %hu\n", seconds ) ;

}

**108. Define three functions—fun1( ), fun2( ) and fun3( ). Each function should receive two integers and return a float. Store the addresses of these functions in an array. Call these functions using the addresses stored in the array.**

ANS:

#include <stdio.h>

float fun1(int a, int b) {

return (float)a / b;

}

float fun2(int a, int b) {

return (float)a \* b;

}

float fun3(int a, int b) {

return (float)a + b;

}

int main() {

float (\*func\_array[3])(int, int);

func\_array[0] = fun1;

func\_array[1] = fun2;

func\_array[2] = fun3;

int x = 5, y = 2;

for (int i = 0; i < 3; i++) {

float result = func\_array[i](x, y);

printf("Result of function %d: %.2f\n", i + 1, result);

}

return 0;

}

**109. Define a function which can find average of the arguments passed to it. Note that in different calls the function may receive different number of arguments**

ANS:

#include <stdio.h>

float calculate\_average(int num, ...) {

va\_list arguments;

float sum = 0.0;

int i;

va\_start(arguments, num);

for (i = 0; i < num; i++) {

sum += va\_arg(arguments, double);

}

va\_end(arguments);

if (num == 0) {

return 0.0;

} else {

return sum / num;

}

}

int main() {

printf("Average of 1, 2, 3: %.2f\n", calculate\_average(3, 1, 2, 3));

printf("Average of 5, 10, 15, 20: %.2f\n", calculate\_average(4, 5, 10, 15, 20));

printf("Average of no arguments: %.2f\n", calculate\_average(0));

return 0;

}

**110. Write a program to calculate the sum of the following series:**

**1! 2! + 2! 3! + 3! 4! + 4! 5! + …… + 9! 10!**

ANS:

#include <stdio.h>

long long int calculate\_series\_sum() {

long long int sum = 0, factorial = 1;

for (int i = 1; i <= 10; i++) {

factorial \*= i;

sum += factorial \* (i + 1);

}

return sum;

}

int main() {

long long int series\_sum = calculate\_series\_sum();

printf("Sum of the series: %lld\n", series\_sum);

return 0;

}

**111. Write a program to enter the numbers till the user wants and at the end it should display the count of positive, negative and zeros entered.**

ANS:

#include <stdio.h>

int main() {

int num, positive = 0, negative = 0, zeros = 0;

char choice;

do {

printf("Enter a number (0 to stop): ");

scanf("%d", &num);

if (num > 0) {

positive++;

} else if (num < 0) {

negative++;

} else {

zeros++;

}

printf("Do you want to continue (y/n)? ");

scanf(" %c", &choice);

} while (choice == 'y' || choice == 'Y');

printf("Positive integers: %d\n", positive);

printf("Negative integers: %d\n", negative);

printf("Zeros: %d\n", zeros);

return 0;

}

**112. Write a program to find the range of a set of numbers that are input through the keyboard. Range is the difference between the smallest and biggest number in the list.**

ANS:

#include <stdio.h>

#include <limits.h>

int main() {

int num, count = 0;

int min\_value = INT\_MAX, max\_value = INT\_MIN;

printf("Enter numbers (0 to stop): ");

while (scanf("%d", &num) == 1 && num != 0) {

count++;

if (num < min\_value) {

min\_value = num;

}

if (num > max\_value) {

max\_value = num;

}

}

if (count == 0) {

printf("No numbers were entered.\n");

} else {

printf("Range: %d\n", max\_value - min\_value);

}

return 0;

}

**113. If three integers are entered through the keyboard, write a program to determine whether they form a Pythagorean triplet or not.**

ANS:

#include <stdio.h>

int main() {

int a, b, c;

printf("Enter three integers (a, b, c): ");

scanf("%d %d %d", &a, &b, &c);

a = abs(a);

b = abs(b);

c = abs(c);

if (a > b) {

int temp = a;

a = b;

b = temp;

}

if (b > c) {

int temp = b;

b = c;

c = temp;

}

if (a \* a + b \* b == c \* c) {

printf("The numbers %d, %d, and %d form a Pythagorean triplet.\n", a, b, c);

} else {

printf("The numbers %d, %d, and %d do not form a Pythagorean triplet.\n", a, b, c);

}

return 0;

}

**114. Write a program that prints sizes of all types of chars, ints and reals**

ANS:

#include <stdio.h>

int main() {

printf("Size of char: %zu byte(s)\n", sizeof(char));

printf("Size of int: %zu byte(s)\n", sizeof(int));

printf("Size of float: %zu byte(s)\n", sizeof(float));

printf("Size of double: %zu byte(s)\n", sizeof(double));

return 0;

}

**115. Write a function that receives as parameters, a 1-D array, its size and an integer and returns number of times the integer occurs in the array.**

ANS:

#include <stdio.h>

int count\_occurrences(int arr[], int size, int target) {

Counts the number of times a target integer appears in a 1-D array.

Args:

arr: The 1-D array of integers.

size: The size of the array.

target: The integer to search for.

int count = 0;

for (int i = 0; i < size; i++) {

if (arr[i] == target) {

count++;

}

}

return count;

}

int main() {

int my\_array[] = {1, 2, 3, 2, 1, 4, 2};

int target\_value = 2;

int occurrences = count\_occurrences(my\_array, sizeof(my\_array) / sizeof(my\_array[0]), target\_value);

printf("The number %d occurs %d times in the array.\n", target\_value, occurrences);

return 0;

}

**116. Create an array of pointers containing names of 10 cities. Write a program that sorts the cities in reverse alphabetical order and prints this reversed list.**

ANS:

#include <stdio.h>

#include <string.h>

int compare\_cities(const void \*a, const void \*b) {

const char \*city1 = \*(const char \*\*)a;

const char \*city2 = \*(const char \*\*)b;

return strcmp(city2, city1);

}

int main() {

char \*cities[] = {"London", "Tokyo", "New York", "Paris", "Berlin",

"Rome", "Madrid", "Singapore", "Los Angeles", "Buenos Aires"};

int num\_cities = sizeof(cities) / sizeof(cities[0]);

qsort(cities, num\_cities, sizeof(char \*), compare\_cities);

printf("Cities in reverse alphabetical order:\n");

for (int i = 0; i < num\_cities; i++) {

printf("%s\n", cities[i]);

}

return 0;

}

**117. Declare a structure called student containing his name, age and address. Create and initialize three structure variables. Define a function to which these variables are passed. The function should convert the names into uppercase. Print the resultant structure variables.**

ANS:

#include <stdio.h>

#include <string.h>

struct student {

char name[50];

int age;

char address[100];

};

void convert\_to\_uppercase(struct student \*students, int num\_students) {

for (int i = 0; i < num\_students; i++) {

strupr(students[i].name);

}

}

int main() {

struct student students[3] = {

{"Alice", 20, "123 Main St"},

{"Bob", 22, "456 Elm St"},

{"Charlie", 21, "789 Oak St"}

};

int num\_students = sizeof(students) / sizeof(students[0]);

convert\_to\_uppercase(students, num\_students);

printf("Students (after converting names to uppercase):\n");

for (int i = 0; i < num\_students; i++) {

printf(" Name: %s, Age: %d, Address: %s\n", students[i].name, students[i].age, students[i].address);

}

return 0;

}

**118. Write a program that checks and reports whether sum of elements in the ith row of a 5 x 5 array is equal to sum of elements in ith column.**

ANS:

#include <stdio.h>

int main() {

int arr[ARRAY\_SIZE][ARRAY\_SIZE];

int row\_sum, col\_sum;

int all\_equal = 1;

printf("Enter elements of the 5x5 array:\n");

for (int i = 0; i < ARRAY\_SIZE; i++) {

for (int j = 0; j < ARRAY\_SIZE; j++) {

scanf("%d", &arr[i][j]);

}

}

for (int i = 0; i < ARRAY\_SIZE; i++) {

row\_sum = 0;

col\_sum = 0;

for (int j = 0; j < ARRAY\_SIZE; j++) {

row\_sum += arr[i][j];

col\_sum += arr[j][i];

}

if (row\_sum != col\_sum) {

all\_equal = 0;

break;

}

}

if (all\_equal) {

printf("Yes, the sum of elements in each row is equal to the sum of elements in the corresponding column.\n");

} else {

printf("No, not all rows and columns have equal sums.\n");

}

return 0;

}

**119. Write a program to multiply two integers using bitwise operators.**

ANS:

#include <stdio.h>

int multiply(int a, int b) {

if (a < 0 && b < 0) {

a = -a;

b = -b;

} else if (a > 0 && b < 0) {

a = -a;

}

int result = 0;

while (b > 0) {

if (b & 1) {

result += a;

}

a <<= 1;

b >>= 1;

}

return result;

}

int main() {

int num1, num2;

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

int product = multiply(num1, num2);

printf("Product of %d and %d using bitwise operators: %d\n", num1, num2, product);

return 0;

}

**120. Write a program to count number of words in a given text file.**

ANS:

#include <stdio.h>

int main() {

char filename[100];

int word\_count = 0;

char ch;

printf("Enter the name of the text file: ");

scanf("%s", filename);

FILE \*fptr = fopen(filename, "r");

if (fptr == NULL) {

printf("Error opening file!\n");

return 1;

}

do {

ch = fgetc(fptr);

} while (ch != EOF && (ch == ' ' || ch == '\t' || ch == '\n'));

while ((ch = fgetc(fptr)) != EOF) {

if (ch == ' ' || ch == '\t' || ch == '\n') {

word\_count++;

do {

ch = fgetc(fptr);

} while (ch != EOF && (ch == ' ' || ch == '\t' || ch == '\n'));

}

}

if (ch != EOF) {

word\_count++;

}

fclose(fptr);

printf("Number of words in the file: %d\n", word\_count);

return 0;

}

**121. Write a program that receives a set of numbers as command- line arguments and prints their average.**

ANS:

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

if (argc <= 1) {

printf("Error: Please provide at least one number as a command-line argument.\n");

return 1;

}

double sum = 0.0;

int num\_values = argc - 1;

for (int i = 1; i < argc; i++) {

char \*endptr;

double value = strtod(argv[i], &endptr);

if (\*endptr != '\0') {

printf("Error: Invalid number '%s' provided as argument.\n", argv[i]);

return 1;

}

sum += value;

}

double average = sum / num\_values;

printf("Average of the numbers: %.2lf\n", average);

return 0;

}

**122. Write a program to check whether contents of the two files are same by comparing them on a byte-by-byte basis.**

ANS:

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

if (argc != 3) {

printf("Usage: %s <filename1> <filename2>\n", argv[0]);

return 1;

}

FILE \*fptr1 = fopen(argv[1], "rb");

FILE \*fptr2 = fopen(argv[2], "rb");

if (fptr1 == NULL || fptr2 == NULL) {

printf("Error opening files!\n");

if (fptr1 != NULL) {

fclose(fptr1);

}

if (fptr2 != NULL) {

fclose(fptr2);

}

return 1;

}

long size1 = ftell(fptr1);

fseek(fptr1, 0, SEEK\_END);

long size2 = ftell(fptr2);

fseek(fptr1, 0, SEEK\_SET);

if (size1 != size2) {

printf("Files have different sizes.\n");

fclose(fptr1);

fclose(fptr2);

return 1;

}

unsigned char byte1, byte2;

while (fread(&byte1, sizeof(byte1), 1, fptr1) == 1) {

if (fread(&byte2, sizeof(byte2), 1, fptr2) != 1) {

printf("Error: Unexpected end of file 2.\n");

fclose(fptr1);

fclose(fptr2);

return 1;

}

if (byte1 != byte2) {

printf("Files differ at byte position %ld.\n", ftell(fptr1) - 1);

fclose(fptr1);

fclose(fptr2);

return 1;

}

}

printf("The contents of the two files are identical.\n");

fclose(fptr1);

fclose(fptr2);

return 0;

}

**123. Write a program that defines a function called isalnum( ). The function should receive a string and check if all characters in it are alphabets or digits. If so, it should return a true, otherwise false. Call this function for the following strings:**

**"ABCD1234"**

**"Nagpur – 440010"**

ANS:

#include <stdio.h>

#include <ctype.h>

int isalnum(const char \*str) {

while (\*str) {

if (!isalnum(\*str)) {

return 0;

}

str++;

}

return 1;

}

int main() {

char \*strings[] = {"ABCD1234", "Nagpur – 440010"};

for (int i = 0; i < sizeof(strings) / sizeof(strings[0]); i++) {

if (isalnum(strings[i])) {

printf("%s is alphanumeric\n", strings[i]);

} else {

printf("%s is not alphanumeric\n", strings[i]);

}

}

return 0;

}

**124. Define an enumeration to represent colors red, green, yellow, magenta and brown. Create two variables Apple and Banana of this enum type and assign colors red and yellow to them respectively. Print these color values and indicate what output will they produce.**

ANS:

#include <stdio.h>

enum Color { RED, GREEN, YELLOW, MAGENTA, BROWN };

int main() {

enum Color Apple, Banana;

Apple = RED;

Banana = YELLOW;

printf("Apple: %d\n", Apple);

printf("Banana: %d\n", Banana);

return 0;

}

**125. Define a function called showbits( ) which displays all the bits of an unsigned char that it receives. Call this function for values 45 and 30. Indicate what output will showbits( ) produce for these values?**

ANS:

#include <stdio.h>

void showbits(unsigned char x) {

for (int i = 7; i >= 0; i--) {

printf("%u", (x & (1 << i)) >> i);

}

printf("\n");

}

int main() {

showbits(45);

showbits(30);

return 0;

}

**126. Write a program to generate and print all unique combinations of numbers 1, 2, 3 and 4.**

ANS:

#include <stdio.h>

void generate\_combinations(int data[], int start, int currentSize, int n) {

if (currentSize == n) {

for (int i = 0; i < n; i++) {

printf("%d ", data[i]);

}

printf("\n");

return;

}

for (int i = start; i <= 4; i++) {

data[currentSize] = i;

generate\_combinations(data, i + 1, currentSize + 1, n);

}

}

int main() {

int n = 4;

int data[n];

printf("All unique combinations of 1, 2, 3, and 4:\n");

generate\_combinations(data, 1, 0, n);

return 0;

}

**127. Define an iterative function and a recursive function to print first ten terms of a Fibonacci series. Which of these two functions will run faster and why?**

ANS:

#include <stdio.h>

void fibonacci\_iterative(int n) {

int a = 0, b = 1, nextTerm;

printf("Fibonacci Series (Iterative): ");

for (int i = 1; i <= n; i++) {

printf("%d, ", a);

nextTerm = a + b;

a = b;

b = nextTerm;

}

printf("\n");

}

int fibonacci\_recursive(int n) {

if (n <= 1) {

return n;

} else {

return fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2);

}

}

int main() {

int numTerms = 10;

fibonacci\_iterative(numTerms);

printf("Fibonacci Series (Recursive): ");

for (int i = 1; i <= numTerms; i++) {

printf("%d, ", fibonacci\_recursive(i));

}

printf("\n");

return 0;

}

**128. Create a data structure for storing the following data:**

**Name of document – Leaflet / Flier / Brochure**

**Number of colors – 1 / 2 / 4 / 5**

**Size – Small / Medium / Big**

**Type of paper – Maplitho / Bond / Artcard**

**Number of copies Type of printing – Positive / Negative**

ANS:

#include <stdio.h>

typedef enum { LEAFLET, FLIER, BROCHURE } DocType;

typedef enum { POSITIVE, NEGATIVE } PrintType;

typedef enum { MAPLITHO, BOND, ARTCARD } PaperType;

typedef enum { SMALL, MEDIUM, BIG } Size;

struct Document {

char name[50];

int num\_colors;

Size size;

PaperType paper;

int num\_copies;

PrintType type;

};

**129. For storing names of months in a year, which out of array of strings and array of pointers to strings is preferable**.

ANS:

#include <stdio.h>

int main() {

char months[12][20] = {

"January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"

};

printf("Months of the year:\n");

for (int i = 0; i < 12; i++) {

printf("%d. %s\n", i + 1, months[i]);

return 0;

}

**130. Consider the following structure:**

**struct Flower**

**{**

**char name[ 20 ] ;**

**int color ;**

**int no\_of\_petals ;**

**} ;**

**struct Flower f[ 3 ] ;**

**Write statements to receive values into array f[ ] and print them on the screen.**

ANS:

#include <stdio.h>

int main() {

struct Flower f[3];

for (int i = 0; i < 3; i++) {

printf("Enter details for Flower %d:\n", i + 1);

printf("Name: ");

scanf("%s", f[i].name);

printf("Color (integer value): ");

scanf("%d", &f[i].color);

printf("Number of petals: ");

scanf("%d", &f[i].no\_of\_petals);

}

printf("\nFlower Details:\n");

for (int i = 0; i < 3; i++) {

printf("Flower %d:\n", i + 1);

printf(" Name: %s\n", f[i].name);

printf(" Color: %d\n", f[i].color);

printf(" Number of petals: %d\n", f[i].no\_of\_petals);

}

return 0;

}

**131. Given two matrices A3 X 3 and B3 X 3, define a function that checks whether matrix A is transpose of matrix B.**

ANS:

#include <stdio.h>

#define SIZE 3

int is\_transpose(int A[SIZE][SIZE], int B[SIZE][SIZE]) {

if (SIZE != B[0][SIZE]) {

return 0;

}

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (A[i][j] != B[j][i]) {

return 0;

}

}

}

return 1;

}

int main() {

int A[SIZE][SIZE] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int B[SIZE][SIZE] = {{1, 4, 7}, {2, 5, 8}, {3, 6, 9}};

if (is\_transpose(A, B)) {

printf("Matrix A is the transpose of matrix B.\n");

} else {

printf("Matrix A is not the transpose of matrix B.\n");

}

return 0;

}

**132. Write a code snippet to carry out the following operations:**

**- Open a file ‘records.dat’ in read binary mode.**

**- Skip first 200 bytes from the beginning of the file.**

**- Read next 20 bytes into an array arr[ ].**

ANS:

#include <stdio.h>

int main() {

FILE \*fp;

char arr[20];

fp = fopen("records.dat", "rb");

if (fp == NULL) {

perror("fopen");

return 1;

}

if (fseek(fp, 200, SEEK\_SET) != 0) {

perror("fseek");

fclose(fp);

return 1;

}

if (fread(arr, 1, 20, fp) != 20) {

perror("fread");

fclose(fp);

return 1;

}

fclose(fp);

return 0;

}

**133. Consider the following statements:**

**int a = 20 ;**

**int \*p ;**

**p = &a ;**

**Write statements only using p to:**

**set a value 45 in a.**

**multiply a with 40 and store the result in a**

**print current value of a**

**Also write statements to perform the following operations:**

**Increment p**

**After incrementation, what will be present in p if variable a is at location 4004?**

**Does incrementing p cause a memory leak?**

ANS:

#include <stdio.h>

int main() {

int a = 20;

int \*p = &a;

\*p = 45;

\*p \*= 40;

printf("Current value of a (using p): %d\n", \*p);

p++;

printf("Value in p after increment (pointing to): %p\n", p);

printf("Memory leak due to incrementing p: No\n");

return 0;

}

**134. Write a program that defines a function called isalpha( ). The function should receive a string and check if all characters in it are alphabets. If so, it should return a true, otherwise false. Call this function for the following strings:**

**"NambyPamby"**

**"Mumbai – 400010"**

ANS:

#include <stdio.h>

#include <ctype.h>

int isalpha(const char \*str) {

while (\*str) {

if (!isalpha(\*str)) {

return 0;

}

str++;

}

return 1;

}

int main() {

char str1[] = "NambyPamby";

char str2[] = "Mumbai – 400010";

if (isalpha(str1)) {

printf("'%s' is all alphabets.\n", str1);

} else {

printf("'%s' contains non-alphabetic characters.\n", str1);

}

if (isalpha(str2)) {

printf("'%s' is all alphabets.\n", str2);

} else {

printf("'%s' contains non-alphabetic characters.\n", str2);

}

return 0;

}

**135. Define an enumeration to represent marital status of a person—single, married, divorced.**

**Create two variables he and she of this enum type and assign to them values single and married**

**respectively. Print these values and indicate what output will they produce.**

ANS:

#include <stdio.h>

enum MaritalStatus { SINGLE, MARRIED, DIVORCED };

int main() {

enum MaritalStatus he, she;

he = SINGLE;

she = MARRIED;

printf("Marital status of he: %s\n", he == SINGLE ? "Single" : (he == MARRIED ? "Married" : "Divorced"));

printf("Marital status of she: %s\n", she == SINGLE ? "Single" : (she == MARRIED ? "Married" : "Divorced"));

return 0;

}

**136. Define functions countzeros( ) and countones( ) which count number of 0s and 1s in an unsigned char that they receive. Call both these functions for values 101 and 111. Indicate what values will these functions return?**

ANS:

#include <stdio.h>

int countzeros(unsigned char uch) {

int count = 0;

for (int bit = 0; bit < 8; bit++) {

if ((uch & (1 << bit)) == 0) {

count++;

}

}

return count;

}

int countones(unsigned char uch) {

int count = 0;

for (int bit = 0; bit < 8; bit++) {

if (uch & (1 << bit)) {

count++;

}

}

return count;

}

int main() {

unsigned char value1 = 101;

unsigned char value2 = 111;

int zeros\_in\_101 = countzeros(value1);

int ones\_in\_101 = countones(value1);

int zeros\_in\_111 = countzeros(value2);

int ones\_in\_111 = countones(value2);

printf("Number of zeros in 101 (binary equivalent): %d\n", zeros\_in\_101);

printf("Number of ones in 101 (binary equivalent): %d\n", ones\_in\_101);

printf("Number of zeros in 111 (binary equivalent): %d\n", zeros\_in\_111);

printf("Number of ones in 111 (binary equivalent): %d\n", ones\_in\_111);

return 0;

}

**137. Write a program to find maximum out of three given numbers in a single statement.**

**What are the pros and cons of using this statement?**

ANS:

#include <stdio.h>

int main() {

int num1, num2, num3, max;

printf("Enter three numbers: ");

scanf("%d %d %d", &num1, &num2, &num3);

max = (num1 > num2) ? (num1 > num3 ? num1 : num3) : (num2 > num3 ? num2 : num3);

printf("Maximum of %d, %d, and %d is: %d\n", num1, num2, num3, max);

return 0;

}

**138. Dynamically allocate space for a 3-D array of dimensions 3 x 5 x 4. Set up each elements of this array with a value 10. Report an error, if enough memory space is not available.**

ANS:

#include <stdio.h>

#include <stdlib.h>

int main() {

const int num\_rows = 3;

const int num\_cols = 5;

const int num\_layers = 4;

int \*\*\*array3d;

array3d = (int \*\*\*)malloc(num\_rows \* sizeof(int \*\*));

if (array3d == NULL) {

fprintf(stderr, "Error: Failed to allocate memory for the first dimension.\n");

return 1;

}

for (int i = 0; i < num\_rows; i++) {

array3d[i] = (int \*\*)malloc(num\_cols \* sizeof(int \*));

if (array3d[i] == NULL) {

fprintf(stderr, "Error: Failed to allocate memory for row %d.\n", i + 1);

for (int j = 0; j < i; j++) {

free(array3d[j]);

}

free(array3d);

return 1;

}

}

for (int i = 0; i < num\_rows; i++) {

for (int j = 0; j < num\_cols; j++) {

array3d[i][j] = (int \*)malloc(num\_layers \* sizeof(int));

if (array3d[i][j] == NULL) {

fprintf(stderr, "Error: Failed to allocate memory for element at row %d, column %d.\n", i + 1, j + 1);

for (int k = 0; k < j; k++) {

free(array3d[i][k]);

}

for (int k = 0; k < i; k++) {

free(array3d[k]);

}

free(array3d);

return 1;

}

for (int k = 0; k < num\_layers; k++) {

array3d[i][j][k] = 10;

}

}

}

printf("Element at (1, 2, 3): %d\n", array3d[1][2][3]);

for (int i = 0; i < num\_rows; i++) {

for (int j = 0; j < num\_cols; j++) {

free(array3d[i][j]);

}

free(array3d[i]);

}

free(array3d);

printf("Memory freed successfully.\n");

return 0;

}

**139. Create an array of pointers to strings for storing names of 5 persons. What is the limitation of this array?**

ANS:

#include <stdio.h>

#include <stdlib.h>

int main() {

const int num\_persons = 5;

char \*names[num\_persons];

names[0] = malloc(sizeof("Alice") + 1);

strcpy(names[0], "Alice");

names[0] = "Alice";

names[1] = "Bob";

for (int i = 0; i < num\_persons; i++) {

printf("Person %d: %s\n", i + 1, names[i]);

}

for (int i = 0; i < num\_persons; i++) {

free(names[i]);

}

return 0;

}

**140. Create a data structure for storing following data:**

**Name of the fruit Color of the fruit**

**Diameter of the fruit Price of the fruit**

**Weight of the fruit**

**Price of the fruit**

**Weight of the fruit**

ANS:

#include <stdio.h>

struct Fruit {

char name[50];

char color[20];

float diameter;

float price;

float weight;

};

int main() {

struct Fruit apple;

strcpy(apple.name, "Apple");

strcpy(apple.color, "Red");

apple.diameter = 7.5;

apple.price = 2.50;

apple.weight = 150.0;

printf("Fruit: %s (%s)\n", apple.name, apple.color);

printf("Diameter: %.2f cm\n", apple.diameter);

printf("Price: %.2f per unit\n", apple.price);

printf("Weight: %.2f grams\n", apple.weight);

return 0;

}

**141. If a function is to be called, is it necessary to mention its prototype declaration?**

ANS:

#include <stdio.h>

int addNumbers(int a, int b);

int main() {

int num1, num2, sum;

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

sum = addNumbers(num1, num2);

printf("Sum of %d and %d is: %d\n", num1, num2, sum);

return 0;

}

int addNumbers(int a, int b) {

return a + b;

}

**142. For a file being pointed to by FILE \*fp, write function calls for carrying out the following operations?**

**Set pointer at 5th position from beginning of file.**

**Set pointer at 20th position from current position.**

**Set pointer at 15th position before end of file.**

ANS:

#include <stdio.h>

int main() {

FILE \*fp = fopen("your\_file.txt", "r");

if (fp == NULL) {

perror("Error opening file");

return 1;

}

int seek\_result = fseek(fp, 4L, SEEK\_SET);

if (seek\_result != 0) {

perror("Error seeking to beginning");

fclose(fp);

return 1;

}

char buffer[10];

fread(buffer, 1, sizeof(buffer), fp);

printf("Read 10 bytes from 5th position: %s\n", buffer);

fseek(fp, 20L, SEEK\_CUR);

if (seek\_result != 0) {

perror("Error seeking from current position");

fclose(fp);

return 1;

}

fread(buffer, 1, sizeof(buffer), fp);

printf("Read 10 bytes from 20th position relative to previous read: %s\n", buffer);

long int file\_size = ftell(fp);

fseek(fp, -15L, SEEK\_END);

if (ftell(fp) < 0) {

fprintf(stderr, "Warning: Could not seek 15 bytes before end (reached beginning of file)\n");

fseek(fp, 0L, SEEK\_SET);

}

fread(buffer, 1, sizeof(buffer), fp);

printf("Read 10 bytes from 15th position before end of file: %s\n", buffer);

fclose(fp);

return 0;

}

**143. In a two-dimensional array a[ 4 ][ 4 ], why do expressions a and \*a yield same base address.**

ANS:

#include <stdio.h>

int main() {

int a[4][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}};

printf("Base address of a: %p\n", a);

printf("Value at a (using a): %d\n", a[0][0]);

printf("Value at a (using \*a): %d\n", \*a);

return 0;

}

**144. Write a program to multiply any two 3 x 3 matrices.**

ANS:

#include <stdio.h>

int main() {

int matrix1[3][3], matrix2[3][3], result[3][3];

int i, j, k;

printf("Enter elements for the first matrix (row-wise):\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

scanf("%d", &matrix1[i][j]);

}

}

printf("Enter elements for the second matrix (row-wise):\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

scanf("%d", &matrix2[i][j]);

}

}

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

result[i][j] = 0;

for (k = 0; k < 3; k++) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

printf("Product of the matrices:\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

printf("%d ", result[i][j]);

}

printf("\n");

}

return 0;

}

**145. Given an array p[ 5 ], write a function to shift it circularly left by two positions. Thus, if the original array is { 15, 30, 28, 19, 61 } then after shifting it will be { 28, 19, 61, 15, 30 } Call this function for a 4 x 5 matrix and get its rows left shifted.**

ANS:

#include <stdio.h>

void shift\_array\_left(int arr[], int n, int d) {

int temp[n];

int i;

for (i = 0; i < d; i++) {

temp[i] = arr[i];

}

for (i = d; i < n; i++) {

arr[i - d] = arr[i];

}

for (i = 0; i < d; i++) {

arr[n - d + i] = temp[i];

}

}

void shift\_matrix\_rows\_left(int matrix[], int rows, int cols, int d) {

for (int i = 0; i < rows; i++) {

shift\_array\_left(matrix + i \* cols, cols, d);

}

}

int main() {

int arr[] = {15, 30, 28, 19, 61};

int n = sizeof(arr) / sizeof(arr[0]);

int d = 2;

shift\_array\_left(arr, n, d);

printf("Shifted array: ");

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

int matrix[4][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15}, {16, 17, 18, 19, 20}};

rows = 4;

cols = 5;

d = 1;

shift\_matrix\_rows\_left((int \*)matrix, rows, cols, d);

printf("Shifted matrix:\n");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

return 0;

}

**146. If the string "Alice in wonder land" is fed to the following scanf( )statement,**

**what will be the contents of arrays str1, str2, str3 and str4?**

**scanf ( "%s%s%s%s", str1, str2, str3, str4 )**

ANS:

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

fgets(str, 100, stdin);

char \*token;

token = strtok(str, " ");

while (token != NULL) {

printf("%s\n", token);

token = strtok(NULL, " ");

}

return 0;

}

**147. To uniquely identify a book a 10-digit ISBN (International Standard Book Number) is used.**

**The rightmost digit in ISBN is a checksum digit. This digit is determined from the other 9 digits using the condition that d1 + 2d2 + 3d3 + ... + 10d10 must be a multiple of 11 (where di denotes the ith digit from the right). The checksum digit d1can be any value from 0 to 10: the ISBN convention is to use the value X to denote 10.**

**Write a program that receives a 10-digit integer, computes the checksum, and reports whether the ISBN number is correct or not.**

ANS:

#include <stdio.h>

int main() {

long long int isbn;

int i, digit, sum = 0, checksum;

printf("Enter a 10-digit ISBN number: ");

scanf("%lld", &isbn);

if (isbn < 1000000000 || isbn > 9999999999) {

printf("Invalid ISBN: Must have exactly 10 digits.\n");

return 1;

}

for (i = 0; i < 10; i++) {

digit = isbn % 10;

isbn /= 10;

if (digit == 'X' || digit == 'x') {

digit = 10;

} else if (digit < 0 || digit > 9) {

printf("Invalid ISBN: Contains non-numeric characters.\n");

return 1;

}

sum += (i + 1) \* digit;

}

checksum = sum % 11;

if (checksum == 0) {

printf("ISBN is valid.\n");

} else {

printf("ISBN is invalid.\n");

}

return 0;

}

**148. A Credit Card number is usually a 16-digit number. A valid Credit Card number would satisfy**

**a rule explained below with the help of a dummy Credit Card number—4567 1234 5678 9129.**

**Start with the rightmost - 1 digit and multiply every other digit by 2.**

**4 5 6 7 1 2 3 4 5 6 7 8 9 1 2 9**

**8 12 2 6 10 14 18 4**

**Then subtract 9 from numbers that are larger than 10. Thus, we get:**

**8 3 2 6 1 5 9 4**

**Add them all up to get 38.**

**Add all the other digits (5, 7, 2, 4, 6, 8, 1, 9) to get 42.**

**Sum of 38 and 42 is 80. Since 80 is divisible by 10, the Credit Card**

**number is valid.**

**Write a program that receives a Credit Card number and checks**

**using the above rule whether the Credit Card number is valid.**

ANS:

#include <stdio.h>

#include <stdbool.h>

bool is\_valid\_credit\_card(long long int card\_number) {

int sum = 0;

bool is\_even\_place = false;

while (card\_number > 0) {

int digit = card\_number % 10;

card\_number /= 10;

if (is\_even\_place) {

digit \*= 2;

}

sum += digit;

is\_even\_place = !is\_even\_place;

}

return sum % 10 == 0;

}

int main() {

long long int card\_number;

printf("Enter a credit card number: ");

scanf("%lld", &card\_number);

if (is\_valid\_credit\_card(card\_number)) {

printf("The credit card number is valid.\n");

} else {

printf("The credit card number is invalid.\n");

}

return 0;

}

**149. How many bytes in memory would be occupied by the following**

**array of pointers to strings? How many bytes would be required to**

**store the same strings in a two-dimensional character array?**

**char \*mess[ ] = {**

**"Hammer and tongs", "Tooth and nail",**

**"Spit and polish", "You and C"**

**} ;**

ANS:

#include <stdio.h>

#include <string.h>

int main() {

char \*mess[] = {"Hammer and tongs", "Tooth and nail", "Spit and polish", "You and C"};

int num\_strings = sizeof(mess) / sizeof(mess[0]);

int pointer\_size = sizeof(char \*);

int pointer\_array\_usage = pointer\_size \* num\_strings;

int max\_string\_length = 15;

int char\_array\_usage = num\_strings \* (max\_string\_length + 1);

printf("Memory usage of the array of pointers to strings: %d bytes\n", pointer\_array\_usage);

printf("Memory usage of the two-dimensional character array: %d bytes\n", char\_array\_usage);

return 0;

}

**150. Write a program to delete all vowels from a sentence. Assume that the sentence is not more than 80 characters long.**

ANS:

#include <stdio.h>

#include <ctype.h>

int main() {

char sentence[81];

int i, j;

printf("Enter a sentence (less than 80 characters): ");

fgets(sentence, 81, stdin);

if (sentence[strlen(sentence) - 1] == '\n') {

sentence[strlen(sentence) - 1] = '\0';

}

for (i = 0; sentence[i] != '\0'; i++) {

sentence[i] = tolower(sentence[i]);

}

i = j = 0;

while (sentence[i] != '\0') {

if (strchr("aeiou", sentence[i]) == NULL) {

sentence[j++] = sentence[i];

}

i++;

}

sentence[j] = '\0';

printf("Sentence without vowels: %s\n", sentence);

return 0;

}