**A42 - Provide a menu to manipulate or display the value of variable of type t**

/\*

Name : Nestin Gregorios Sunny

Date : .2025

Description :

A menu to manipulate or display the value of variable of type t

Allocate 8 consecutive bytes in memory

Provide a display menu

1. Add element

2. Remove element

3. Display element

4. Exit from the program

It should allow to add elements of different data types which data type size is less than or equal to 8

\*/

#include<stdio.h>

#include<stdlib.h>

char ch1\_flag;

char ch2\_flag;

short short\_flag;

int int\_flag;

float float\_flag;

double double\_flag;

void add\_element(void \*ptr);

void remove\_element(void \*ptr);

void display\_element(void \*ptr);

int main()

{

void \*ptr;

ptr = (void \*)malloc(8); //Dynamic Memory Allocation

int choice;

do

{

printf("\nMenu: \n"); //Display Menu

printf("1. Add element\n");

printf("2. Remove element\n");

printf("3. Display element\n");

printf("4. Exit from the program\n");

int choice;

printf("\nChoice ---> "); //Read Choice from user

scanf("%d", &choice);

switch(choice)

{

case 1:

add\_element(ptr);

break;

case 2:

remove\_element(ptr);

break;

case 3:

display\_element(ptr);

break;

case 4:

printf("Exiting...\n");

exit(0);

break;

default:

printf("Invalid choice..");

}

} while (choice != 4);

return 0;

}

void add\_element(void \*ptr)

{

printf("\nEnter the type you have to insert: \n"); //Display the menu

printf("1.int\n");

printf("2.char\n");

printf("3.short\n");

printf("4.float\n");

printf("5.double\n");

int choice;

printf("choice ---> "); //Read choice from user

scanf("%d", &choice);

if(choice == 1)

{

if(double\_flag == 0 && float\_flag == 0 && int\_flag == 0) //Checking for possibility

{

printf("Enter the int: ");

scanf("%d", (int \*)ptr + 1); //Before storing integer value to the void pointer we must dereference first to int pointer

int\_flag = 1; //Updating the flag to 1

}

else{

printf("Don't have enough space to store\n"); //Error message

}

}

else if(choice == 2)

{

if(double\_flag == 0)

{

if(ch1\_flag == 0)

{

printf("Enter the char: ");

scanf(" %c", (char \*)ptr + 0); //Before storing character value to the void pointer we must dereference first to char pointer

ch1\_flag = 1; //Updating the flag to 1

}

else if(ch2\_flag == 0)

{

printf("Enter the char: "); //Before storing character value to the void pointer we must dereference first to char pointer

scanf(" %c", (char \*)ptr + 1);

ch2\_flag = 1;

}

else{

printf("Don't have enough space to store\n");

}

}

else{

printf("Don't have enough space to store\n");

}

}

else if(choice == 3)

{

if(double\_flag == 0 && short\_flag == 0)

{

printf("Enter the short: ");

scanf("%hd", (short \*)ptr + 1); //Before storing short value to the void pointer we must dereference first to short pointer

short\_flag = 1; //Updating the flag to 1

}

else{

printf("Don't have enough space to store\n");

}

}

else if(choice == 4)

{

if(double\_flag == 0 && float\_flag == 0 && int\_flag == 0)

{

printf("Enter the float: ");

scanf("%f", (float \*)ptr + 1); //Before storing float value to the void pointer we must dereference first to float pointer

float\_flag = 1; //Updating the flag to 1

}

else{

printf("Don't have enough space to store\n");

}

}

else if(choice == 5)

{

if(double\_flag == 0)

{

if(int\_flag == 1 || ch1\_flag == 1 || ch2\_flag == 1 || short\_flag == 1 || float\_flag == 1)

{

printf("Don't have enough space to store\n");

}

else{

printf("Enter the double: ");

scanf("%lf", (double \*)ptr + 0); //Before storing double value to the void pointer we must dereference first to double pointer

double\_flag = 1; // Updating the flag to 1

}

}

else{

printf("Don't have enough space to store\n");

}

}

else{

printf("Ivalid Choice, Enter only digits(1 - 5)..\n");

}

}

void remove\_element(void \*ptr)

{

display\_element(ptr);

int index;

printf("Enter the index value to be deleted: ");

scanf("%d", &index);

switch(index) //Removing the element by updating flag value to 0

{

case 0:

ch1\_flag = 0;

double\_flag = 0;

break;

case 1:

ch2\_flag = 0;

break;

case 2:

short\_flag = 0;

break;

case 4:

int\_flag = 0;

float\_flag = 0;

break;

default:

printf("Invalid input\n");

return;

}

printf("index %d successfully deleted\n", index);

}

void display\_element(void \*ptr)

{

int i = 0;

printf("--------------------\n");

if(ch1\_flag == 1)

{

printf("0 -> %c (char)\n", \*((char \*)ptr + 0)); //To get value we must dereference

}

if(ch2\_flag == 1)

{

printf("1 -> %c (char)\n", \*((char \*)ptr + 1));

}

if(short\_flag == 1)

{

printf("2 -> %hd (short)\n", \*((short \*)ptr + 1));

}

if(int\_flag == 1)

{

printf("4 -> %d (int)\n", \*((int \*)ptr + 1));

}

if(float\_flag == 1)

{

printf("4 -> %f (float)\n", \*((float \*)ptr + 1));

}

if(double\_flag == 1)

{

printf("0 -> %lf (double)\n", \*((double \*)ptr + 0));

}

printf("--------------------\n");

}

**A45 - WAP to generate a n\*n magic square**

/\*

Name : Nestin Gregorios Sunny

Date : 09.07.2025

Description :

Create a magic square

Sample Input :

Enter a number : 3

Sample Output :

8 1 6

3 5 7

4 9 2

\*/

#include <stdio.h>

#include<stdlib.h>

void magic\_square(int \*\*, int);

int main()

{

int n, i, j;

//printf("Enter a number : ");

scanf("%d", &n);

if((n % 2 == 0) || (n < 0))

{

printf("Error : Please enter only positive odd numbers\n");

return 0;

}

//Dynamic memory allocation

int \*\*ptr;

ptr = (int \*\*)calloc(n, sizeof (int \*));

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

{

ptr[i] = calloc(n, sizeof (int));

}

magic\_square(ptr, n);

//print magic number

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

{

for(j = 0; j < n; j++)

{

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

}

printf("\n");

}

//free allocated memory

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

{

free(ptr[i]);

}

free(ptr);

return 0;

}

void magic\_square(int \*\*arr, int n)

{

int i, row = 0, col = n/2; //start from 1st row, middle column

int pre\_row, pre\_col;

for(i = 1; i <= n\*n; i++)

{

if(arr[row][col] != 0)

{

//move back to previous column

row = pre\_row;

col = pre\_col;

//down one row

row++;

if(row == n)

{

row = 0;

}

}

arr[row][col] = i;

//save current index of row and column

pre\_row = row;

pre\_col = col;

//one row up and one column right

row--;

col++;

//if pointer goes out of loop (min index value is 0 and max index value is n - 1)

if(row == -1) //invalid index

{

row = n - 1; //moving to last row

}

if(col == n)

{

col = 0; //moving to 1st column

}

}

}

**A43 - Variance calculation with dynamic arrays**

/\*

Name : Nestin Gregorios Sunny

Date : 09.07.2025

Description :

Variance calculation with dynamic array

Sample Input :

Enter the no. of elements : 5

Enter 5 elements :

[0] -> 1

[1] -> 2

[2] -> 3

[3] -> 4

[4] -> 5

Sample Output :

Variance is 2.000000

\*/

#include <stdio.h>

#include<stdlib.h>

float variance(int \*, int);

void read\_arr(int \*, int);

int main()

{

int i, n;

printf("Enter the no. of elements : ");

scanf("%d", &n);

int \*ptr = malloc(n \* sizeof(int));

printf("Enter the %d elements :\n",n);

read\_arr(ptr, n);

//find varaiance

float var = variance(ptr, n);

printf("Variance is %f\n", var);

free(ptr);

return 0;

}

void read\_arr(int arr[], int n)

{

int i;

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

{

printf("[%d] -> ", i);

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

}

}

float variance(int arr[], int n)

{

int i, sum = 0, dsum = 0;

float mean, var;

//find mean of array

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

{

sum += arr[i];

}

mean = (float) sum/n;

//find deviation and squared deviation

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

{

arr[i] = arr[i] - mean;

arr[i] = arr[i] \* arr[i];

dsum += arr[i];

}

var = (float) dsum / n;

return var;

}

**A46 - WAP to implement fragments using Array of Pointers**

#include <stdio.h>

#include<stdlib.h>

void fragments(int, int \*, float \*[]);

void print\_avg(int, int \*,float \*[]);

void sort\_avg(int, int \*,float \*[]);

int main()

{

int i, j, r;

//Read no. of rows

printf("Enter no. of rows : ");

scanf("%d", &r);

//Declare the array of pointer

float \*arr[r];

//Read no. of columns for each row

int col\_arr[r];

for(i = 0; i < r; i++)

{

printf("Enter the no of columns in rows[%d] :", i);

scanf("%d", &col\_arr[i]);

}

//Allocate the memory for columns + 1(avg)

float \*\*a = malloc( r \* sizeof (float\*));

for(i = 0; i < r; i++)

{

a[i] = malloc((col\_arr[i] + 1) \* sizeof(float));

}

//Read elements for each rows

for(i = 0; i < r; i++)

{

printf("Enter %d values for row[%d] : ", col\_arr[i], i);

for(j = 0; j < col\_arr[i]; j++)

{

scanf("%f", &a[i][j]);

}

}

fragments(r, col\_arr, a);

printf("\nBefore sorting output is :\n");

print\_avg(r, col\_arr, a);

sort\_avg(r, col\_arr, a);

printf("\nAfter sorting output is :\n");

print\_avg(r, col\_arr, a);

//free array of pointer

for(i = 0; i < r; i++)

{

free(a[i]);

}

free(a);

return 0;

}

void fragments(int row, int col\_arr[], float \*a[])

{

int i, j;

for(i = 0; i < row; i++)

{

float sum = 0;

for(j = 0; j < col\_arr[i]; j++)

{

sum += a[i][j];

}

float avg = sum / col\_arr[i];

a[i][col\_arr[i]] = avg;

col\_arr[i] += 1;

}

}

void print\_avg(int row, int col\_arr[],float \*a[])

{

int i, j;

for(i = 0; i < row; i++)

{

for(j = 0; j < col\_arr[i]; j++)

{

printf("%f ", a[i][j]);

}

printf("\n");

}

}

void sort\_avg(int row, int col\_arr[],float \*a[])

{

int i, j;

for(i = 0; i < row - 1; i++)

{

for(j = i + 1; j < row; j++)

{

//last element is the avaerage of that particular row

float avg\_i = a[i][col\_arr[i] - 1];

float avg\_j = a[j][col\_arr[j] - 1];

if(avg\_i > avg\_j)

{

float \*temp = a[i];

a[i] = a[j];

a[j] = temp;

int temp\_col = col\_arr[i];

col\_arr[i] = col\_arr[j];

col\_arr[j] = temp\_col;

}

}

}

}

**A44 - WAP to convert Little Endian data to Big Endian**

/\*

Name : Nestin Gregorios Sunny

Date : 10.07.2025

Description :

Convert little endian data to big endian

Sample Input :

Enter the size: 2

Enter any number in hexadecimal : ABCD

Sample Output :

After conversion CDAB

\*/

#include<stdio.h>

#include<stdlib.h>

void swap\_bytes(void \*ptr, int size);

int main()

{

int size;

printf("Enter the size : ");

scanf("%d", &size); //either 2 or 4

if(size == 2)

{

short int num;

printf("Enter any number in hexadecimal : ");

scanf("%hx", &num);

swap\_bytes(&num, size);

printf("After conversion : %hx\n", num);

}

else if(size == 4)

{

int num;

printf("Enter any number in hexadecimal : ");

scanf("%x", &num);

swap\_bytes(&num, size);

printf("After conversion : %08X\n", num);

}

else

{

printf("Invalid size\n");

}

return 0;

}

void swap\_bytes(void \*ptr, int size)

{

char \*bytes = (char \*)ptr;

int start = 0, end = size - 1;

while(start < end)

{

char temp = bytes[start];

bytes[start] = bytes[end];

bytes[end] = temp;

start++;

end--;

}

}

**A47 - Read n & n person names of maxlen 20. Sort and print the names**

/\*

Name : Nestin Gregorios Sunny

Date : 11.07.2025

Description :

Sort and print the names in alphabetical order

Sample Input :

Enter the size: 5

Enter the 5 names of length max 20 characters in each

[0] -> Delhi

[1] -> Agra

[2] -> Kolkata

[3] -> Bengaluru

[4] -> Chennai

Sample Output :

The sorted names are:

Agra

Bengaluru

Chennai

Delhi

Kolkata

\*/

#include <stdio.h>

#include<string.h>

#include<stdlib.h>

//function declaration

void my\_strcpy(char \*, const char \*);

int my\_strcmp(const char \*, const char \*);

void sort\_names(char (\*)[20], int);

int main()

{

int i, row;

//printf("Enter the size :");

scanf("%d", &row);

char (\*ptr)[20];

ptr = malloc(row \* sizeof(\*ptr) );

for(i = 0; i < row; i++)

{

//printf("[%d] -> ", i);

scanf(" %s", ptr[i]);

}

sort\_names(ptr, row); //sort function call

printf("The sorted names are :\n");

for(i = 0; i < row; i++)

{

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

}

//free pointer

// for(i = 0; i < row; i++)

// {

// free(ptr[i]);

// }

// free(ptr);

return 0;

}

//function definitions

void sort\_names(char (\*ptr)[20], int row)

{

int i, j;

char temp[20];

for(i = 0; i < row - 1; i++)

{

for(j = 0; j < row - i - 1; j++)

{

if(my\_strcmp(ptr[j], ptr[j + 1]) > 0)

{

my\_strcpy(temp, ptr[j]);

my\_strcpy(ptr[j], ptr[j + 1]);

my\_strcpy(ptr[j + 1], temp);

}

}

}

}

int my\_strcmp(const char \*src, const char \*dest)

{

while(\*src != '\0' && \*dest != '\0')

{

if(\*src != \*dest)

{

return \*src - \*dest;

}

src++;

dest++;

}

return \*src - \*dest;

}

void my\_strcpy(char \*dest, const char \*src)

{

while(\*dest++ = \*src++);

}

**A48 - WAP to find the product of given matrix.**

/\*

Name : Nestin Gregorios Sunny

Date : 11.07.2025

Description :

Find product of given matrix

Sample Input :

Enter number of rows : 3

Enter number of columns : 3

Enter values for 3 x 3 matrix :

1 2 3

1 2 3

1 2 3

Enter number of rows : 3

Enter number of columns : 3

Enter values for 3 x 3 matrix :

1 1 1

2 2 2

3 3 3

Sample Output :

Product of two matrix :

14 14 14

14 14 14

14 14 14

\*/

#include <stdio.h>

#include <stdlib.h>

int matrix\_mul(int \*\*, int, int, int \*\*, int, int, int \*\*);

int main()

{

int i, j;

int \*\*mat\_a, \*\*mat\_b, \*\*result;

int r1, c1, r2, c2;

printf("Enter number of rows : ");

scanf("%d", &r1);

printf("Enter number of columns : ");

scanf("%d", &c1);

mat\_a = malloc(r1 \* sizeof(int \*));

for(i = 0; i < r1; i++)

{

mat\_a[i] = malloc(c1 \* sizeof(int));

}

printf("Enter values for %d x %d matrix :", r1, c1);

for(i = 0; i < r1; i++)

{

for(j = 0; j < c1; j++)

{

scanf("%d", &mat\_a[i][j]);

}

}

printf("Enter number of rows : ");

scanf("%d", &r2);

printf("Enter number of columns : ");

scanf("%d", &c2);

if(c1 != r2)

{

printf("Matrix multiplication is not possible\n");

return -1;

}

mat\_b = malloc(r2 \* sizeof(int \*));

for(i = 0; i < r2; i++)

{

mat\_b[i] = malloc(c2 \* sizeof(int));

}

printf("Enter values for %d x %d matrix :", r2, c2);

for(i = 0; i < r2; i++)

{

for(j = 0; j < c2; j++)

{

scanf("%d", &mat\_b[i][j]);

}

}

result = malloc(c1 \* sizeof(int\*));

for(i = 0; i < c1; i++)

{

result[i] = malloc(r2 \* sizeof(int));

}

matrix\_mul(mat\_a, r1, c1, mat\_b, r2, c2, result);

printf("Product of two matrix :\n");

for(i = 0; i < r1; i++)

{

for(j = 0; j < c2; j++)

{

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

}

printf("\n");

}

//free memory

for(i = 0; i < r1; i++)

{

free(mat\_a[i]);

free(result[i]);

}

free(mat\_a);

free(result);

for(i = 0; i < r2; i++)

{

free(mat\_b[i]);

}

free(mat\_b);

return 0;

}

int matrix\_mul(int \*\*m1, int r1, int c1, int \*\*m2, int r2, int c2, int \*\*result)

{

int i, j, k;

for(i = 0; i < r1; i++)

{

for(j = 0; j < c2; j++)

{

result[i][j] = 0;

for(k = 0; k < c1; k++)

{

result[i][j] += m1[i][k] \* m2[k][j];

}

}

}

return 1;

}