

G V V Sharma*

Problem 1. Write a C program to generate an arithmetic progression (AP) with first term $a = -1$, last term $l = 1$ and number of terms $n = 100$. Store these numbers in a pointer array.

Solution:

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

//Main function
int main(void)
{
// Variable declarations
double a = -1.0, l = 1.0, d, *ap;
int n = 100, i;

// Creating memory for ap
ap = (double *)malloc(n * sizeof(
double));

//Common difference
d = (l-a)/(n-1);

// Generating the AP array
for(i = 0; i < 100; i++)
{
ap[i] = a+i*d;
}

// Printing values
for(i = 0; i < n; i++)
printf("%lf\n", ap[i]);

free(ap);
return 0;
}
```

Problem 2. Modify the above program to create a function for generating the AP pointer array.

Solution:

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

double *linspace_pointer(double ,
double , int );
int main(void)
{
double a = -1.0, l = 1.0, *ap;
int n = 100, i;

// Assigning pointer to a
ap = linspace_pointer(a,l,n);

for(i = 0; i < n; i++)
printf("%lf\n", ap[i]);

//Common difference

return 0;
}

double *linspace_pointer(double a,
double l, int n)
{
// Variable declarations
double d, *ap;
int i;

// Creating memory for ap
ap = (double *)malloc(n * sizeof(
double));

//Common difference
d = (l-a)/(n-1);

// Generating the AP
```

*The author is with the Department of Electrical Engineering, Indian Institute of Technology, Hyderabad 502205 India e-mail: gadepall@iith.ac.in. All material in this document is released under GNU GPL. Free to use for anything

```

for(i = 0; i < 100; i++)
{
ap[i] = a+i*d;
}
//Returning the address of the
  first memory block
return ap;
}

```

Problem 3. Repeat the above exercise through a list.

Solution:

```

#include <stdio.h>
#include <stdlib.h>

typedef struct list
{
double data;
struct list *next;
}node;

node *linspace_pointer(double ,
  double , int );

int main(void)
{
node *ap;
double a = -1.0, l = 1.0;
int n = 100;

// Getting the head of the AP list
ap = linspace_pointer(a,l,n);

// Printing the AP
while (ap->next != NULL)
{
printf("%lf\n", ap->data);
ap = ap->next;
}

return 0;
}

node *linspace_pointer(double a ,
  double l , int n)
{
// Variable declarations
node *ap, *head;

```

```

double d;
int i;

//Common difference
d = (l-a)/(n-1);

ap = (node *) malloc (sizeof(node));
head = ap;
// Generating the AP
for(i = 0; i < 100; i++)
{

ap->data = a+i*d;
// Creating memory for next node
ap->next = (node *) malloc (sizeof(
  node));
// Initializing next node
ap->next->next = NULL;
//node increment
ap = ap->next;
}
//Returning the address of the
  first memory block
return head;
}

```

Consider the polynomials

$$p(x) = x + 1 \quad (3.1)$$

$$q(x) = x^2 + 2x + 3 \quad (3.2)$$

Problem 4. Polynomial Addition: Evaluate $p(x) + q(x)$ using pointer arrays.

Problem 5. Repeat the above exercise using a list.

Problem 6. Polynomial Multiplication: Using convolution, find $p(x)q(x)$ using pointer arrays

Problem 7. Repeat the above exercise using a list.

Problem 8. Generalize the above polynomial operations for any degree using both pointer arrays and lists.

Problem 9. Matrix Operations: Create a matrix using pointer arrays

Solution:

```

#include <stdio.h>
#include <stdlib.h>

```

```
// This program shows how to use
// pointers as 2-D arrays

// Function declaration
double **createMat(int m, int n);
void readMat(int m, int n, double **
    p);
void print(int m, int n, double **p)
    ;
// End function declaration

int main() // main function begins
{

// Defining the variables
int m, n; // integers
double **a;

printf("Enter the size of the
    matrix m n\n");
scanf("%d %d", &m, &n);

printf("Enter the values of the
    matrix\n");
a = createMat(m, n); // creating the
    matrix a
readMat(m, n, a); // reading values
    into the matrix a
print(m, n, a); // printing the matrix
    a

return 0;
}

// Defining the function for matrix
// creation
double **createMat(int m, int n)
{
    int i;
    double **a;

// Allocate memory to the pointer
a = (double **) malloc(m * sizeof(
    *a));
    for (i=0; i<m; i++)
        a[i] = (double *) malloc(n
            * sizeof(*a[i]));
```

```
return a;
}
// End function for matrix creation

// Defining the function for
// reading matrix
void readMat(int m, int n, double **
    p)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%lf", &p[i][j]);
        }
    }
}
// End function for reading matrix

// Defining the function for
// printing
void print(int m, int n, double **p)
{
    int i, j;

    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
            printf("%lf ", p[i][j]);
        printf("\n");
    }
}
```

Problem 10. Let

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \quad (10.1)$$

Use pointer arrays for the following.

- 1) Generate A^t which is the *transpose* of A .
- 2) Obtain $A + A^t$.
- 3) Obtain $A - A^t$.
- 4) Obtain AA^t .
- 5) Obtain A^{-1} .

Problem 11. Repeat the above exercise using a two dimensional list.

Problem 12. Binary Search Tree:

- 1) Enter the list of numbers

$$S = \{3, 6, 2, 1, 5, 9, 4, 7, 0, 8\} \quad (12.1)$$

into a binary tree in such a fashion that the smaller number goes to the left brach.

- 2) Access this tree in such a manner as to print the numbers in ascending order.
- 3) Repeat the exercise to print the numbers in descending order.
- 4) Try to do both the above exercises using recursion.

Problem 13. Polynomial Division: Let

$$q(x) = p(x)g(x) + r(x), \quad (13.1)$$

where $g(x)$ is the quotient polynomial and $r(x)$ is the remainder polynomial. Obtain the coefficient list for $g(x)$ and $r(x)$.