20CYS181 C -PROGRAMMING LAB

CB.EN.U4CYS22047

EXERCISE: 1ND MATRIX MULTIPLICATION FIXED

AIM: Program is to multiply two matrix with fixed number of rows and Column

CODE:

```
#include<stdio.h>
int main(){
   int a [3][3] ,b[3][3];
    int c[3][3] = {{0,0,0}, {0,0,0}, {0,0,0}};
    int i, j, k;
    for(i=0;i<3;i++){
       for(j=0;j<3;j++){
          scanf("%d",&a[i][j]);
       }
   }
    for(i=0;i<3;i++){
       for(j=0;j<3;j++){
          scanf("%d",&b[i][j]);
       }
   }
```

```
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            for(k=0;k<3;k++){
                c[i][j] += a[i][k]*b[k][j];
            }
        }
    }
    for(i=0;i<3;i++){
         for(j=0;j<3;j++){
            printf("%d \t",c[i][j]);
        }
        printf("\n");

    }
 return 0;
}
```

ALGORITHM:

1. Start
2. Declare and initialize matrices a, b, and c.
3. Initialize variables i, j, and k.
4. Set each element of matrix c to zero.
5. Read values for matrix a from the user.
6. Read values for matrix b from the user.
7. Perform matrix multiplication:

a. For i = 0 to the number of rows in matrix a

i. For j = 0 to the number of columns in matrix b

i. Set c[i][j] to zero

ii. For k = 0 to the number of columns in matrix a

a. Set c[i][j] to c[i][j] + a[i][k] * b[k][j]

iii. End For

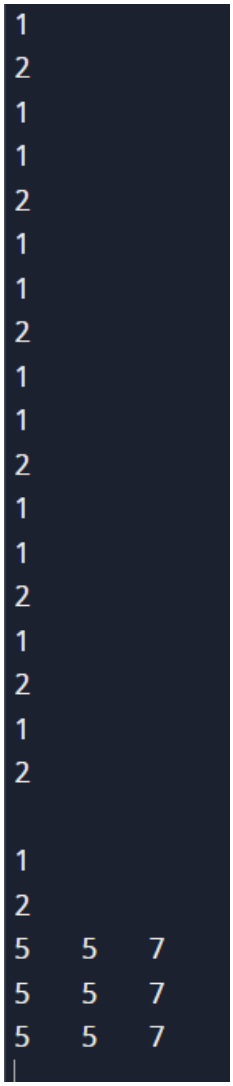
ii. End For

b. End For

8. Display the result matrix c.

9. End

OUTPUT:



Exercise2:MATRIX MULTIPLICATION VARIABLE

CODE:

```
#include<stdio.h>
int main(){
    int i,j,k;
    int m,n;
    scanf("%d",&m);
    scanf("%d",&n);
```

```
int a[100][100],b[100][100],c[100][100];

for(i=0;i<m;i++){
   for(j=0;j<n;j++){
       scanf("%d",&a[i][j]);
   }
}

for(i=0;i<m;i++){
    for(j=0;j<n;j++){
       scanf("%d",&b[i][j]);
   }
}

for(i=0;i<m;i++){
   for(j=0;j<n;j++){
       c[i][j]=0;
      for(k=0;k<n;k++){
          c[i][j] += a[i][k]*b[k][j];
      }
   }
}
 for(i=0;i<m;i++){
   for(j=0;j<n;j++){
```

```
            printf("%d \t",c[i][j]);
        }
         printf("\n");
    }
     return 0;
}
```

ALGORITHM:

Step 1: Start the main program

Step 2: Declare variables i, j, k, m, and n as integers

Step 3: Read the values of m and n from the user, where m represents the number of rows and n represents the number of columns

Step 4: Declare three 2-dimensional arrays: a, b, and c

Step 5: Initialize the outer loop i from 0 to m-1

- 5.1: Inside the i loop, initialize the inner loop j from 0 to n-1
  - 5.1.1: Read the values from the user and store them in a[i][j]
  - 5.1.2: End the inner loop
- 5.2: End the outer loop

Step 6: Initialize the outer loop i from 0 to m-1

- 6.1: Inside the i loop, initialize the inner loop j from 0 to n-1
  - 6.1.1: Read the values from the user and store them in b[i][j]
  - 6.1.2: End the inner loop
- 6.2: End the outer loop

Step 7: Initialize the outer loop i from 0 to m-1

Step 8: Inside the i loop, initialize the inner loop j from 0 to n-1

Step 9: Inside the j loop, initialize the variable k from 0 to n-1

9.1: Calculate the product of a[i][k] and b[k][j]

9.2: Add the product to c[i][j]

9.3: End the k loop

Step 10: End the j loop

Step 11: End the i loop

Step 12: Initialize the outer loop i from 0 to m-1

12.1: Inside the i loop, initialize the inner loop j from 0 to n-1

12.1.1: Display the value of c[i][j]

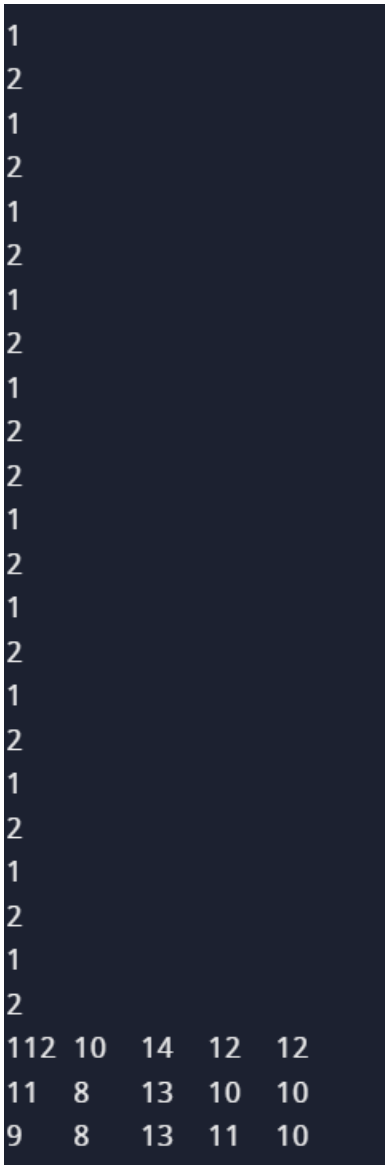
12.1.2: End the inner loop

12.2: Display a newline character

12.3: End the outer loop

Step 13: End the main program

OUTPUT:



Exercise3:Half Piramid

Code: #include<stdio.h>

```
int main()
{
   int row, i, j;
```

```
    scanf("%d", &row);

    if(row <= 0 || row >= 25)
    {
        printf("Invalid Input\n");
    }
    else
            for(i = 0; i < row; i++)
        {
            for(j = 0; j <= i; j++)
            {
                printf("*\t");
            }
            printf("\n");
        }
    }
    return 0;
}
```

Algorithm:

Step 1: Start the main program

Step 2: Declare variables row, i, and j as integers

Step 3: Read the value of row from the user

Step 4: Check if the input is valid by using an if statement

4.1: If row <= 0 or row >= 25, print "Invalid Input"

4.2: Else, continue to the next step

Step 5: Use nested loops to generate the pattern of asterisks

5.1: Outer loop: Initialize i from 0 to row-1

5.1.1: Inner loop: Initialize j from 0 to i

5.1.1.1: Print an asterisk followed by a tab ("\t")

5.1.2: Print a newline character ("\n") to move to the next row

Step 6: End

Output:

Exercise4:Full Pyramid

Code: #include<stdio.h>

```
int main()
{
   int rows, i, j, space;
   scanf("%d", &rows);
   if (rows > 25 || rows < 0)
```

```
{
   printf("Invalid Input");
}
else
{

   for (i = 1; i <= rows; i++)

      for (space = 1; space <= rows - i; space++)
      {
         printf("  ");
      }


      for (j = 1; j <= 2 * i - 1; j++)
      {
         printf("* ");
      }

      printf("\n");
   }
}

return 0;
```

}

Algorithm:

Step 1: Start

Step 2: Declare variables rows, i, j, and space as integers

Step 3: Read the value of rows from the user

Step 4: Check if the input is valid by using an if statement

4.1: If rows is greater than 25 or less than 0, print "Invalid Input"

4.2: Else, continue to the next step

Step 5: Use a for loop to iterate from i = 1 to i = rows

5.1: Inside the loop, use another for loop to print spaces

5.1.1: Initialize space = 1, and continue the loop until space <= rows - i

5.1.2: Print two spaces (" ")

5.2: Inside the loop, use another for loop to print asterisks

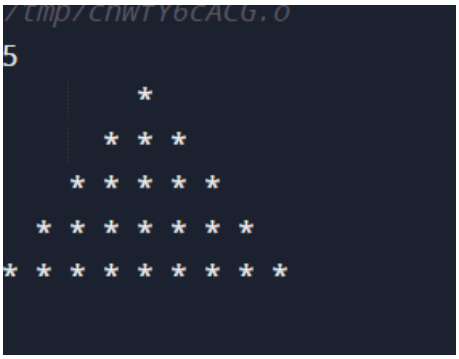
5.2.1: Initialize j = 1, and continue the loop until j <= 2 * i - 1

5.2.2: Print an asterisk followed by a space ("* ")

5.3: Print a newline character ("\n") to move to the next line

Step 6: End

Output:



Exercise5:Palindromic Pattern Half Pyramid

Code:

```
#include <stdio.h>
int main()
{
        int i, j, rows;
        scanf("%d", &rows);
        printf("*\n");
        for(i=1; i<=rows; i++)
        {
            for(j=1; j<=i; j++)
                    {
         if(j==1){printf("*");}
          printf("%d",j);
                    }
            for(j=i-1; j>=1; j--)
                    {
          printf("%d",j);


                    }
        printf("*\n");
        }
          for(i=rows-1;i>=1;i--)
```

```
    {
            printf("*");
            for(j=1;j<=i;j++)
            {printf("%d",j);}
            for(j=i-1;j>=1;j--)
            {printf("%d",j);}
            printf("*\n");

    }
        printf("*\n");
        return 0;
}
```

Algorithm:

1. Start
2. Declare variables: i, j, rows
3. Read the value of 'rows' from the user
4. Print a single asterisk followed by a newline
5. // Upper half of the pattern
   5.1. Loop from i = 1 to i <= rows
      5.1.1. Print an asterisk
      5.1.2. Loop from j = 1 to j <= i
         5.1.2.1. If j is equal to 1, then print an asterisk
         5.1.2.2. Print the value of j

5.1.3. Loop from j = i-1 to j >= 1

5.1.3.1. Print the value of j

5.1.4. Print an asterisk followed by a newline

6. // Lower half of the pattern

6.1. Loop from i = rows-1 to i >= 1

6.1.1. Print an asterisk

6.1.2. Loop from j = 1 to j <= i

6.1.2.1. Print the value of j

6.1.3. Loop from j = i-1 to j >= 1

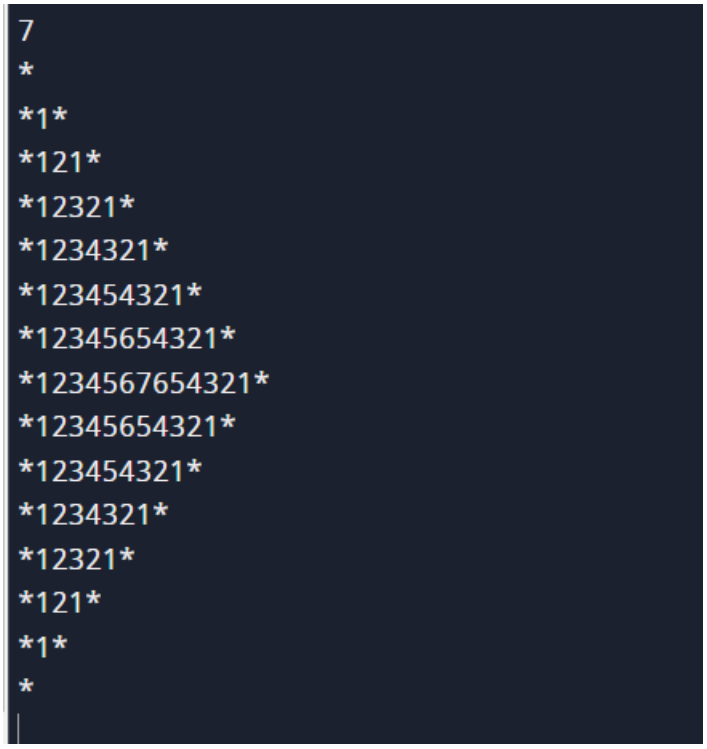
6.1.3.1. Print the value of j

6.1.4. Print an asterisk followed by a newline

7. Print a single asterisk followed by a newline

8. End

Output:



Exercise6:Palindrome Check

Code:

```
#include<stdio.h>


int main()
{
   int n, n1, rev = 0;
   scanf("%d", &n);
   n1 = n;
   if (n >= 0)
   {
      while (n > 0)
      {
         int temp = n % 10;
```

```
        rev = rev * 10 + temp;
        n = n / 10;
    }

    if (rev == n1)
    {
        printf("Is a palindrome.");
    }
    else {
        printf("Is not a palindrome.");
    }
}
else
{
    n1 = n1 * -1; // Convert it to positive
    int n2=n1;
    while (n1 > 0) // Find the reverse of the number
    {
        int temp = n1 % 10;
        rev = rev * 10 + temp;
        n1 = n1 / 10;
    }
    if (rev == n2)
    {
```

```
            printf("Is a palindrome.");
        }
        else {
            printf("Is not a palindrome.");
        }
    }
    return 0;
}
```

ALGORITHM:

STEP1: declare varibles n and n1 and rev=0

Step2: read input from user and store in n

Step3: n1=n

Step4: initialize rev=0

Step5: if n >= 0

Step6: while n>0

Step7: declare temp and assign n%10

Step8: assign rev=rev*10+temp

Step9: assign n equal to n/10

Step10: if (rev ==n1)

10.1: display is a palindrome

10.2: is not a palindrome

Step11: otherwise n is a negative

Step12: n1=n1*-1

Step13: declare n2=n1

Step14: while n1>0

14.1: assign n1= n1%10

14.2: read n1 into temp

14.3: update rev = rev*10+temp

14.4: assign n1=n1/10

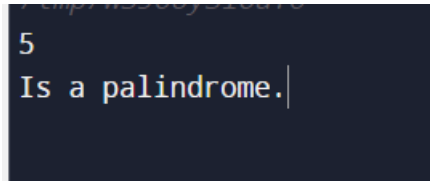
Step15: if rev is equal to n2

Step 16: display is a palindrome

Step17: otherwise is not a palindrome

Step18: end

Output:



Exercise7:Addition and subtraction of complex using structure

Code:

```
#include<stdio.h>

#define size 2

struct complex {
```

```
    int real;
    int complex;
};

int main()
{
    struct complex c[size];
    int j, i, res1, res2, re1, re2;
        for(j = 0; j < size; j++)
        {
            printf("Enter the real part of complex number %d: ", j + 1);
            scanf("%d", &c[j].real);
            printf("Enter the complex part of complex number %d: ", j + 1);
            scanf("%d", &c[j].complex);
        }
    res1 = c[0].real + c[1].real;
    res2 = c[0].complex + c[1].complex;
    re1 = c[0].real - c[1].real;
    re2 = c[0].complex - c[1].complex;
    printf("Addition: %d+%di\n", res1, res2);
    printf("Subtraction: %d+%di\n", re1, re2);

    return 0;
}
```

Algorithm:

step1. Start

step2. Declare constant size=2.

step3. Declare a structure complex with members

3.1. Integer variable real.

3.2. Integer variable complex.

step4. Declare a structure complex c of size size.

step5. Declare variables j,I,res1,res2,re1,re2.

step6. Initialise j=0

6.1. Display “Enter the real part of complex number” & value of j+1 & ”:”.

6.2. Read the value of c[j].real.

6.3. Display “Enter the complex part of complex number” & value of j+1 & “:”.

6.4. Read the value of c[j].complex.

step7. Iterate until j<size

step8. Assign res1 = c[0].real + c[1].real.

step9. Assign res2 = c[0].complex + c[1].complex.

step10. Assign re1 = c[0].real - c[1].real.

step11. Assign re2 = c[0].complex - c[1].complex.

step12. Display “Addition: ” &value of res1 & “+” & value of res2 &”i” and go to next line.

step13. Display “Subtraction: “ & value of re1 & “+” & value of re2 & “i” and go to next line.

step14. End.

Output:

```
Enter the real part of complex number 1: 5
Enter the complex part of complex number 1: 4
Enter the real part of complex number 2: 7
Enter the complex part of complex number 2: 3
Addition: 12+7i
Subtraction: -2+1i
```

Exercise8:

Calculate Electricty Consumption

Code:

```
#include <stdio.h>
void main()
{
   int ar_ebno[5],ar_units[5],ebno,units,ar_fee[5],fee;
   char ar_name[5],name;
   for(int i=0; i<=4; i++)
   {
      printf("please enter the name\n");
      scanf("%s", &name);
      ar_name[i]=name;
      printf("please enter the eb number\n");
      scanf("%d", &ebno);
      ar_ebno[i]=ebno;
```

```
printf("enter the units consumed\n");
scanf("%d", &units);
ar_units[i]=units;
if (units<100 && units>0)
{
   printf("the cost is free free\n");
   fee=0;
   fee=ar_fee[i];
}
else if (units>100 && units<400)
{
   fee=2.25*units;
   ar_fee[i]=fee;
}
else if (units>400 && units<500)
{
  fee=4*units;
   ar_fee[i]=fee;
}
else if (units>500 && units<600)
{
   fee=6*units;
   ar_fee[i]=fee;
}
```

```
        else if (units>600)
        {
            fee=8*units;
            ar_fee[i]=fee;
        }
        else if (units<0)
        {
            printf("units cant be negative\n");
        }
        else
        {
            printf("invalid input\n");
        }

    }

    for(int j=0; j<=4; j++)
    {

    printf("%d %d %d \n",ar_ebno[j],ar_units[j],ar_fee[j]);
    }
}
```

Algorithm:

1. Start

2. Declare variables:

- ar_ebno: an integer array to store the eb numbers
- ar_units: an integer array to store the consumed units
- ebno, units: integer variables to hold the current eb number and units
- ar_fee: an integer array to store the calculated fees
- fee: integer variable to hold the current fee
- ar_name: a character array to store the names
- name: character variable to hold the current name

3. Start a loop from i = 0 to i <= 4 to get input and calculate fees for each customer

3.1. Print "Please enter the name"

3.2. Read the name into the variable 'name'

3.3. Store the name in the 'ar_name' array at index 'i'

3.4. Print "Please enter the eb number"

3.5. Read the eb number into the variable 'ebno'

3.6. Store the eb number in the 'ar_ebno' array at index 'i'

3.7. Print "Enter the units consumed"

3.8. Read the units consumed into the variable 'units'

3.9. Store the units consumed in the 'ar_units' array at index 'i'

3.10. Check the value of 'units' to determine the fee:

- If 'units' is less than 100 and greater than 0:
    - Print "The cost is free free"
    - Set 'fee' to 0
    - Store 'fee' in the 'ar_fee' array at index 'i'
- Else if 'units' is between 100 and 400:
    - Calculate 'fee' as 2.25 multiplied by 'units'
    - Store 'fee' in the 'ar_fee' array at index 'i'
- Else if 'units' is between 400 and 500:
    - Calculate 'fee' as 4 multiplied by 'units'
    - Store 'fee' in the 'ar_fee' array at index 'i'
- Else if 'units' is between 500 and 600:
    - Calculate 'fee' as 6 multiplied by 'units'
    - Store 'fee' in the 'ar_fee' array at index 'i'
- Else if 'units' is greater than 600:
    - Calculate 'fee' as 8 multiplied by 'units'
    - Store 'fee' in the 'ar_fee' array at index 'i'
- Else if 'units' is less than 0:
    - Print "Units can't be negative"
- Else:
    - Print "Invalid input"

4. Print the results for each customer:

4.1. Start a loop from j = 0 to j <= 4

4.1.1. Print the eb number, consumed units, and fee stored in 'ar_ebno[j]', 'ar_units[j]', and 'ar_fee[j]' respectively

5. End

Output:

```
please enter the name
Vamsi
please enter the eb number
3456
enter the units consumed
5
the cost is free free
```

Exercise9:Number Diamond Pattern

Code:

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

void printDiamond(int n) {
   int i, j, k;

   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n - i; j++) {
         printf("  ");
```

```
        }

        for (k = 9; k >= 9 - (2 * i - 2); k--) {
            printf("%d ", k);
        }

        printf("\n");
    }

    for (i = n - 1; i >= 1; i--) {
        for (j = 1; j <= n - i; j++) {
            printf("  ");
        }

        for (k = 9; k >= 9 - (2 * i - 2); k--) {
            printf("%d ", k);
        }

        printf("\n");
    }
}

int main() {
```

```
    int n;

    scanf("%d", &n);

    printDiamond(n);

    return 0;
}
```

Algorithm:

(i) Declare a void FUNCTION printDiamond with parameter as an integer variable n

STEP1. Start

STEP2. Declare variables i,j,k.

STEP3. Initialise i=1

3.1. Initialise j=1

3.1.1. Display " ".

3.2. Iterate until j=n-i.

3.3. Initialise k=9(decreasing)

3.3.1. Display the value of k.

3.4. Iterate until k>=9-(2*i-2)

3.5. Go to the next line.

STEP4. Iterate until i=n

STEP5. Initialise i=n-1 (decreasing)

5.1. Initialise j=1

5.1.1. Display " ".

5.2. Iterate until j=n-i

5.3. Initialise k=9(decreasing)

5.3.1. Display the value of k

5.4. Iterate until k = 9 - (2 * i - 2)

5.5. Go to next line.

STEP6. Iterate until i=1

STEP7. End.

(ii) MAIN

STEP8. Start.

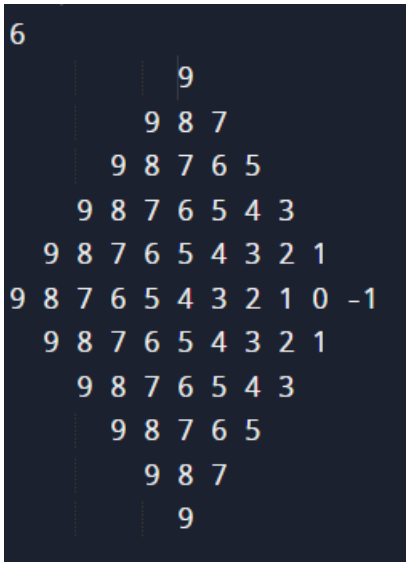
STEP9. Declare variable n.

STEP10. Read the value of n.

STEP11. Call the function printDiamond with parameter as n.

STEP12. End.

Output:



Exercise10:Alphabet Hourglass pattern

Code:

```
#include <stdio.h>


int main()
{
  int rows, i, j, k;
  scanf("%d", &rows);
  rows = rows+1;
  for(i=rows-1;i>1;i--){
    for(k=0;k<rows-i;k++) {
        printf("  ");
    }
    int alphabet = 68;
    for(j=0;j<2*i-1;j++){
```

```
        printf("%c ",alphabet);
        ++alphabet;
    }
    if(i>2){
    printf("\n");
    }
  }
  for(i=0;i<rows;i++){
    for(k=0;k<rows-i;k++) {
        printf("  ");
    }
    int alphabet = 68;
    for(j=1;j<=2*i-1;j++){
        if(j>=1){
            printf("%c ",alphabet);
        ++alphabet;
        }
    }
    printf("\n");
  }
  return 0;

}
```

Algorithm:

1.Start.

2.Declare integer variables rows, i, j, k.

3.Read the number of rows as input to ‘rows’.

4.Increment rows by 1.

5.Initiate a for loop (decreasing) with i = rows - 1.

5.1.Initiate a for loop with k = 0.

5.1.1.Display “ ”(two spaces).

5.1.2.Repeat from 5.1.1 upto k = rows-i.

5.2.Declare an integer variable “alphabet” assigned with value of 68.

5.3.Initiate a for loop with j = 0.

5.3.1.Display character of ascii value “alphabet”.Increment alphabet.

5.3.2.Repeat 5.3.1 upto j = 2*i.

5.4.If ‘i’ is greater than 2, go to new line. Otherwise, continue.

5.5 – Repeat from 5.1 upto i = 1.

6.Initiate a for loop (increasing) with i = rows - 1.

6.1.Initiate a for loop with k = 0.

6.1.1.Display “ ”(two spaces).

6.1.2.Repeat from 6.1.1 upto k = rows-i.

6.2.Declare an integer variable “alphabet” assigned with value of 68.

6.3.Initiate a for loop with j = 0.

6.3.1.Display character of ascii value “alphabet”. Increment alphabet.

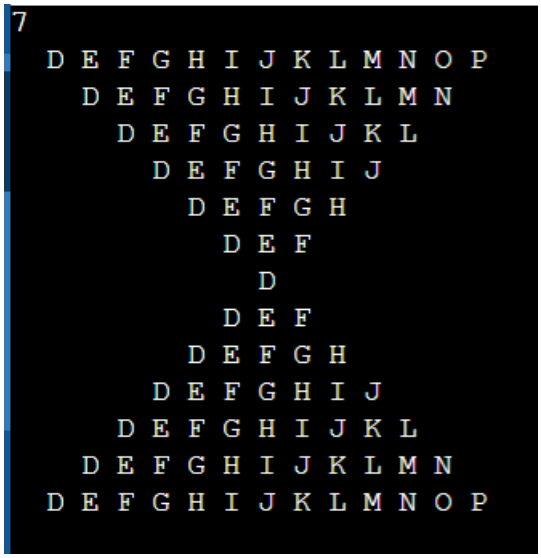
6.3.2.Repeat 6.3.1 upto j = 2*i.

6.4 – Go to new line.

6.5 – Repeat from 5.1 upto i = 1.

7.End

Output:



Exercise11:Generate All The Prime Number Between Two Given Numbers

Code:

```
#include<stdio.h>

int isprime(int num)

{

   int i, check = 0;
```

```
    for(i = 1; i <= num; i++)
    {
        if(num % i == 0)
        {
            check++;
        }
    }
    if(check == 2)
    {
        printf("%d ", num);
    }
}
void generateprime(int start, int end)
{
    int num;
    printf("Prime numbers between %d and %d are: ", start, end);
    for(num = start; num <= end; num++)
    {
        isprime(num);//Calling the isprime function
    }
}

int main()
{
```

```
    int start, end;
    scanf("%d%d", &start, &end);
    if(start>0&&end>0)
    {
    generateprime(start, end
    }
    else{
        printf("The given range is negative");
    }
    return 0;
}
```

ALGORITHAM:

(i) Declare an integer FUNCTION isprime with parameter as integer variable num.

Step1. Start

step2. Declare variables i, check and initialise check=0;

step3. Initialise i=1

3.1. Check if num%i=0

3.1.1. Then assign check=check+1.

step4. Iterate until i=num.

step5. Check if check=2

5.1. Then Display value of num.

step6. End.

(ii) Declare a void FUNCTION generateprime with parameters as integer variables start and end.

step1. Start.

step2. Declare variable num.

step3. Display “Prime numbers between “ & value of start & “and” & value of end & “are:”.

step4. Initialise num=start

4.1. Call function isprime with parameter as num.

step5. End.

(iii) MAIN:

step1. Start.

step2. Declare variables start, end.

step3. Read the values of start and end.

step4. Check if start>0 and end>0

4.1. Then call function generateprime with parameters as start and end.

step5. Else

5.1. Display “The given range is negative”.

step6. End.

Output:

```
25 50
Prime numbers between 25 and 50 are: 29 31 37 41 43 47
```

Exercise12:

AREA USING STRUCT, ENUM, AND UNION

CODE:

```
#include<stdio.h>
#define PI 3.14
enum Type{
Circle=1,
Rectangle };
union dimensions{
struct circle{
float radius; }c;
struct rectangle{
float radius;
float length;
}r;
};
struct Shape{
union dimensions d;
enum Type type;
};
int main(){
struct Shape area;
float k;
scanf("%d",&area.type);
```

```
switch(area.type){
case Circle:
scanf("%f",&area.d.c.radius);
k = PI * area.d.c.radius * area.d.c.radius;
printf("Area of the circle: %.4f units",k);
break;
case Rectangle:
scanf("%f",&area.d.r.radius);
scanf("%f",&area.d.r.length);
k = area.d.r.radius * area.d.r.length;
printf("Area of the rectangle: %.4f units",k);
break;
default:
printf("Invalid choice!");
}
return 0;
}
```

ALGORITHM:

1)Start the program.

2)Define the constant ‘PI’ as 3.14 using ‘#define’.

3)Define the enum ‘Type’ and the union ‘dimensions’.

4)Define the struct ‘shape’ that contains the unions ‘dimensions’ and the enum ‘Type’.

5)Declare the variable ‘area’ of type ‘Shape’.

6)Read the value of ‘area.type’.

7)Use a ‘switch’ statement to perform actions based on the selected shape.

8)Caluculate the area based on the selected shape.

9)Print the caluculated area with the appropriate shape label.

10)End of the program.

OUTPUT:

Exercise13:Employe Management System

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

enum EmployeeType {
    FullTime,
    PartTime
};

union EmployeeDetails {
```

```
    struct {
        float monthlySalary;
        float bonus;
    } fullTime;
    struct {
        float hourlyRate;
        int hoursWorked;
    } partTime;
};

struct Employee {
    char name[50];
    int age;
    enum EmployeeType type;
    union EmployeeDetails details;
};

int main() {
    int numEmployees;
    printf("Enter the number of employees: ");
    scanf("%d", &numEmployees);
    struct Employee *employees = malloc(numEmployees *
sizeof(struct Employee));
```

```
for (int i = 0; i < numEmployees; i++) {
    printf("\nEnter details for Employee %d\n", i + 1);
    printf("Name: ");
    scanf("%s", employees[i].name);
    printf("Age: ");
    scanf("%d", &employees[i].age);

    int employeeType;
    printf("Employee Type (0 for Full-time, 1 for Part-time): ");
    scanf("%d", &employeeType);

    if (employeeType == FullTime) {
        employees[i].type = FullTime;
        printf("Monthly Salary: ");
        scanf("%f", &employees[i].details.fullTime.monthlySalary);
        printf("Bonus: ");
        scanf("%f", &employees[i].details.fullTime.bonus);
    } else if (employeeType == PartTime) {
        employees[i].type = PartTime;
        printf("Hourly Rate: ");
        scanf("%f", &employees[i].details.partTime.hourlyRate);
        printf("Hours Worked: ");
        scanf("%d", &employees[i].details.partTime.hoursWorked);
    } else {
```

```
            printf("Invalid employee type! Setting as Full-time.\n");
            employees[i].type = FullTime;
            printf("Monthly Salary: ");
            scanf("%f", &employees[i].details.fullTime.monthlySalary);
            printf("Bonus: ");
            scanf("%f", &employees[i].details.fullTime.bonus);
        }
    }

    int choice;

    printf("\nEmployee Management System\n");
    printf("1. Calculate total monthly salary for all full-time employees\n");
    printf("2. Calculate total earnings for all part-time employees\n");
    printf("3. Display details of all employees\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: {
            float totalMonthlySalary = 0.0;
            for (int i = 0; i < numEmployees; i++) {
```

```
            if (employees[i].type == FullTime) {
                totalMonthlySalary += employees[i].details.fullTime.monthlySalary;
            }
        }
        printf("Total monthly salary for all full-time employees: %.2f\n", totalMonthlySalary);
        break;
    }
    case 2: {
        float totalEarnings = 0.0;
        for (int i = 0; i < numEmployees; i++) {
            if (employees[i].type == PartTime) {
                totalEarnings += employees[i].details.partTime.hourlyRate * employees[i].details.partTime.hoursWorked;
            }
        }
        printf("Total earnings for all part-time employees: %.2f\n", totalEarnings);
        break;
    }
    case 3: {
        printf("\nDetails of all employees:\n");
        for (int i = 0; i < numEmployees; i++) {
            printf("Employee %d\n", i + 1);
```

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

            if (employees[i].type == FullTime) {
                printf("Employee Type: Full-time\n");
                printf("Monthly Salary: %.2f\n",
employees[i].details.fullTime.monthlySalary);
                printf("Bonus: %.2f\n",
employees[i].details.fullTime.bonus);
            } else if (employees[i].type == PartTime) {
                printf("Employee Type: Part-time\n");
                printf("Hourly Rate: %.2f\n",
employees[i].details.partTime.hourlyRate);
                printf("Hours Worked: %d\n",
employees[i].details.partTime.hoursWorked);
            }

            printf("--------------------\n");
        }
        break;
    }
    case 4:
        printf("Exiting the program. Goodbye!\n");
        break;
    default:
```

```
            printf("Invalid choice!\n");
            break;
    }

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

ALGORITHM:

STEP1: start

Step2: declare variable enumEMPLOYEE to store the number of employees

Step3: read the value of enum EMPLOYEE

STEP4: Initialise i=0 to i<enumEMPLOYEE

STEP5:read the details for the employee from user including age name and employee type

STEP6: based on the employee type:

6.1 if the type Is full time:

6.2 read the monthly salary and bonus

6.3if the type is part time:

6.4:read the hourly rate and hours worled from the user

6.5: if the employee type is neither full time nor part time , set it

Full time and read the monthly salary and bonus

Step7: based on choice:

7.1: if the choice is 1(calculate total monthly salary for all full-time employee)

7.2:initialize totalMonthly salary as 0

7.3:for each employee

STEP8: IF THE EMPLOYEE IS PART TIME CALCULATE their earnings by multiplying the hourly rate with the hours worked and add it to TOTALEARNING

STEP9: display totalEarnings

STEP10: if it is 3(Display details of all employees)

STEP11:display the employee details including name,ageand based on the employee type print the corresponding details

STEP12: IF IT IS 4 exit the program

STEP 13: IF it is an invalid choice print an error message

STEP14 :free the allocated memory

STEP15:end

Output:

```
Enter the number of employees: 2
Enter details for Employee 1
Name: vamsi
Age: 19
Employee Type (0 for Full-time, 1 for Part-time): 1
Hourly Rate: 100
Hours Worked: 10
Enter details for Employee 2
Name: kiran
Age: 20
Employee Type (0 for Full-time, 1 for Part-time): 0
Monthly Salary: 50000
Bonus: 5
Employee Management System
1. Calculate total monthly salary for all full-time employees
2. Calculate total earnings for all part-time employees3. Display details of all
    employees
4. Exit
Enter your choice:
```

Example14:Simple Student Management System

CODE:

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

struct students{
    char name[30];
    int age;
    enum score{
        A,B,C,D,F
    }e_s;
}b[72];



int main(){
    char grade[10];
    int val_menu,t=0,error;
    do{
        scanf("%d",&val_menu);

        switch (val_menu){

            case 1:
                if(t==3){
                    printf("Maximum number of students reached.\n");
```

```
}
else{
   scanf("%s",b[t].name);
   scanf("%d",&b[t].age);
   scanf("%s",grade);
   if (strcmp(grade,"A")==0){
      b[t].e_s=0;
   }
   else if (strcmp(grade,"B")==0){
      b[t].e_s=1;
   }
   else if (strcmp(grade,"C")==0){
      b[t].e_s=2;
   }
   else if (strcmp(grade,"D")==0){
      b[t].e_s=3;
   }
   else if (strcmp(grade,"F")==0){
      b[t].e_s=4;
   }
   else{
      b[t].e_s=4;
   }
   t=t+1;
   printf("Student added successfully.\n");
}
```

```
    break;


case 2: //displaying students
    printf("List of students:\n");
    for(int i=0;i<t;i++){
        printf("Student %d\n",i+1);
        printf("Name: %s\n",b[i].name);
        printf("Age: %d\n",b[i].age);
        if (b[i].e_s==0){
            printf("Score: A\n");
        }
        else if (b[i].e_s==1){
            printf("Score: B\n");
        }
        else if (b[i].e_s==2){
            printf("Score: C\n");
        }
        else if (b[i].e_s==3){
            printf("Score: D\n");
        }
        else if (b[i].e_s==4){
            printf("Score: F\n");
        }
        printf("\n");
    }
```

```
    break;



case 3: //higest score
    printf("Highest-scoring student:\n");
    for(int i=0;i<t;i++){
        if (b[i].e_s==0){
            printf("Name: %s\n",b[i].name);
            printf("Age: %d\n",b[i].age);
            if (b[i].e_s==0){
                printf("Score: A\n");
            }
            else if (b[i].e_s==1){
                printf("Score: B\n");
            }
            else if (b[i].e_s==2){
                printf("Score: C\n");
            }
            else if (b[i].e_s==3){
                printf("Score: D\n");
            }
            else if (b[i].e_s==4){
                printf("Score: F\n");
            }
        }
```

```
            break;
        }
        printf("\n");
        break;


    case 4: //exiting
        printf("Exiting the program. Thank you for using our system!");
        break;
    }
  } while (val_menu!=4);
}
```

Algorithm –

1) define enum with name ‘grade’ with elements A,B,C,D,F
2) define structure “student” with elements character string ‘name’ , integer age, enum grade ‘score’
3) define function add student taking arguments of student(structure) , and integer i
4) define function displaystudentdetails taking a structure and integer I as an argument
5) define findhighestscoringstudent() taking a structure ‘s’ and integer I as argument
6) START
7) Declare structure s[MAX_STUDENTS]
8) Declare and assign I = 0
9) Declare variable choice(int)
10) do – read ‘choice’
11) take in the choice for the switch case
    a) Case =1
        i) If I < MAX_STUDENTS

i. call add student with arguments (structure name s and integer i)

addstudent –

1) declare character tempgrade
2) read s[i].name
3) read s[i].age
4) read tempgrade
4.1) if tempgrade = ‘A’
4.1.1) s[i].score = A
4.2) else if tempgrade = ‘B’
4.2.1) s[i].score = B
4.3) else if tempgrade = ‘C’
4.3.1)s[i].score = C
4.4) else if tempgrade = ‘D’
4.4.1) s[i].score = D
4.5) else if tempgrade = ‘F’
4.5.1) s[i].score = F
4.6) otherwise
4.6.1) display “invalid grade”

ii) Otherwise display ‘student limit exceeded’

iii) break

12) end function

B) Case 2

1. If I = 0
   1.1. Display ‘no students to display’
2. Otherwise
   2.1. Initialize I = 0
      2.1.1. Call displaystudentdetails function with arguments structure s , I
   2.2. Iterate till I = MAX_STUDENTS
3. Break;

C) Case 3 –

1) call findhighestscoringstudent with arguments structure s and integer i

Find findhighestscoringstudent

i) declare variable highscore

i) 1. Assign highscore = s[0].score

2. Initialize I = 0

3. if s[i].score <highscore

3.1 assign highscore = i

ii) display ‘ Highest – scoring student’ and move to next line

iii) call displaystudentdetails with parameters structure s and integer highscore

iv) break

D) Case 4 :

I) display “exiting the program . Thank you for using our system! and go to next line

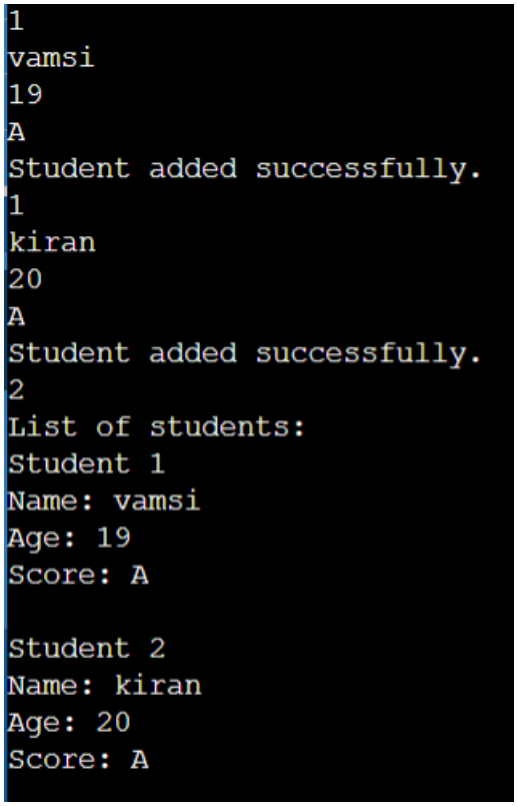
2) End

3) break

E) Default

1) display “invalid choice! Choose again.” And move to the next line

Output:



Exercise15:Decimal Number to Binary,Octal and Hexadecimal

Code:

```
#include <stdio.h>


void binary(int c) {
   int n[50];
   int i = 0;
   while (c> 0) {
      n[i] = c % 2;
      c = c / 2;
      i++;
```

```
    }
    printf("Binary equivalent: ");
    for (int j = i - 1; j >= 0; j--) {
        printf("%d", n[j]);
    }
    printf("\n");
}
void octal(int c) {
    int n[50];
    int i = 0;

    while (c> 0) {
        n[i] = c % 8;
        c = c / 8;
        i++;
    }
    printf("Octal equivalent: ");
    for (int j = i-1 ; j >=0; j--) {
        printf("%d", n[j]);
    }
    printf("\n");
}
void hexadecimal(int c) {
    int n[50];
```

```
int i = 0;
while (c> 0) {
   n[i] = c % 16;
   c = c / 16;
   i++;
}
printf("Hexadecimal equivalent: ");
for (int j = i-1 ; j >= 0; j--) {
   if(n[j]==10){
      printf("A");
   }
   else if(n[j]==11){
      printf("B");
   }
    else if(n[j]==12){
      printf("C");
   }
    else if(n[j]==13){
      printf("D");
   }
    else if(n[j]==14){
      printf("E");
   }
    else if(n[j]==15){
```

```
            printf("F");
        }
        else if (n[j]<10){
        printf("%d", n[j]);
        }
    }
    printf("\n");
}
int main() {
    int c;
    scanf("%d", &c);
if(c<1){
    printf("Error: Value should be greater than 0");
}
else{
    binary(c);
    octal(c);
    hexadecimal(c);
}
    return 0;
}
```

Algorithm:

1. Start
2. Include the necessary header files (`stdio.h`).

3. Define the function `binary(c)`:
   3.1. Declare an integer array `n` with a size of 50.
   3.2. Declare an integer variable `i` and initialize it to 0.
   3.3. Iterate while `c` is greater than 0:
      3.3.1. Set `n[i]` to the remainder of `c` divided by 2 (`c % 2`).
      3.3.2. Set `c` to `c` divided by 2 (`c / 2`).
      3.3.3. Increment `i` by 1.
   3.4. Print "Binary equivalent: ".
   3.5. Iterate from `j` equals `i - 1` to 0:
      3.5.1. Print `n[j]`.
   3.6. Print a new line.
4. Define the function `octal(c)`:
   4.1. Declare an integer array `n` with a size of 50.
   4.2. Declare an integer variable `i` and initialize it to 0.
   4.3. Iterate while `c` is greater than 0:
      4.3.1. Set `n[i]` to the remainder of `c` divided by 8 (`c % 8`).
      4.3.2. Set `c` to `c` divided by 8 (`c / 8`).
      4.3.3. Increment `i` by 1.
   4.4. Print "Octal equivalent: ".
   4.5. Iterate from `j` equals `i - 1` to 0:
      4.5.1. Print `n[j]`.
   4.6. Print a new line.
5. Define the function `hexadecimal(c)`:
   5.1. Declare an integer array `n` with a size of 50.
   5.2. Declare an integer variable `i` and initialize it to 0.
   5.3. Iterate while `c` is greater than 0:
      5.3.1. Set `n[i]` to the remainder of `c` divided by 16 (`c % 16`).
      5.3.2. Set `c` to `c` divided by 16 (`c / 16`).
      5.3.3. Increment `i` by 1.
   5.4. Print "Hexadecimal equivalent: ".
   5.5. Iterate from `j` equals `i - 1` to 0:
      5.5.1. If `n[j]` is equal to 10, print "A".

5.5.2. Else if `n[j]` is equal to 11, print "B".
5.5.3. Else if `n[j]` is equal to 12, print "C".
5.5.4. Else if `n[j]` is equal to 13, print "D".
5.5.5. Else if `n[j]` is equal to 14, print "E".
5.5.6. Else if `n[j]` is equal to 15, print "F".
5.5.7. Else if `n[j]` is less than 10, print `n[j]`.
5.6. Print a new line.
6. Define the `main()` function:
6.1. Declare an integer variable `c`.
6.2. Read an integer value into `c` using `scanf()`.
6.3. If `c` is less than 1:
6.3.1. Print "Error: Value should be greater than 0".
6.4. Else:
6.4.1. Call the function `binary(c)`.
6.4.2. Call the function `octal(c)`.
6.4.3. Call the function `hexadecimal(c)`.
6.5. Return 0.
7. End the program.

Output: