# YPH 205
# SOLUTIONS TO PRESCRIBED C PROGRAMS

This document was typeset using LaTeX. If you are interested, you can download and examine the `.tex` file at `vhbelvadi.com/teaching`, suggest edits and alternate solutions and more. Updated 27.01.2020.

**EXECUTION INSTRUCTIONS**

1. Open Terminal, Command Prompt or equivalent Command Line Interface.
2. Create a new folder with `mkdir <folderName>`. **Use new folders for each program.**
3. Open the new folder with `cd <folderName>` and type `vi <name>.c` to open a new file.
4. Type your program in the file. Use `I` to start typing and `Esc` to stop typing.
5. After typing, hit `Esc` then save the program with `:w` and quit with `:q`, or just use `:wq`.
6. Compile the program with `clang -o <name> <name>.c` or use `gcc` instead of `clang`.
7. Execute the program using `./<name>`.

*In all cases above, replace* `<...>` *with a name of your choice (obviously). Try to avoid spaces.*

## 1. FINDING THE ROOTS OF A QUADRATIC EQUATION

```
1   #include<stdio.h>
2   #include<math.h>
3   #include<stdlib.h>
4
5   int main(){
6
7       float a,b,c;
8       float d,root1,root2;
9
10      printf("Enter a, b and c of quadratic equation ax^2 + bx + c below:
            \n\n");
11      scanf("%f%f%f",&a,&b,&c);
12
13      system("clear");
14
15      d = b * b - 4 * a * c;
16
17      if(d < 0){
18          printf("\nThe equation has complex roots.\n");
19
20          printf("They are ");
21          printf("%.3f+%.3fi",-b/(2*a),sqrt(-d)/(2*a));
```

```
22            printf("and %.3f+%.3fi.\n\n",-b/(2*a),-sqrt(-d)/(2*a));
23
24            return 0;
25        }
26        else if(d==0){
27            printf("The equation has two equal roots.\n");
28
29            root1 = -b /(2* a);
30            printf("They are both %.3f.\n\n",root1);
31
32            return 0;
33        }
34        else{
35            printf("The equation has are real roots.\n");
36
37            root1 = ( -b + sqrt(d)) / (2* a);
38            root2 = ( -b - sqrt(d)) / (2* a);
39            printf("They are %.3f and %.3f.\n\n",root1,root2);
40        }
41
42        return 0;
43    }
```

**COMPILING INSTRUCTIONS**

Since we are using `#include<math.h>` in this program, we need to add a special flag `-lm` to tell the compiler to include the mathematics library. Therefore, for such a program the combined compilation and execution statements would be `clang -o <name> <name>.c -lm && ./<name>`

## 2. LEAST SQUARE FITTING A CO-ORDINATE DATA SET FROM A FILE

**SPECIAL INSTRUCTIONS**

*Before starting this program, create a new folder and follow these instructions:*

1. Create a new text file using `vi input.txt` or equivalent.
2. Enter tab-separated values of your choice, preferably a linear data set with minor errors, e.g.

```
2    5
3    7
4    8
5    11
```

3. Save and quit with `:wq` and, optionally, use `ls` to ensure the file exists.
4. Type and execute your C program in the same folder as usual.

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

int main() {

    FILE *input;
    int n,i,x[20],y[20],sumx=0,sumy=0,sumxy=0,sumx2=0;
    float m,c;
    char name[20];

    system("clear");

    printf("\n\nEnter the name of the input file: \t");
    scanf("%s", name);

    printf("\n\nEnter the number of rows of data:\t");
    scanf("%d",&n);

    input = fopen(name,"r");

    for(i=0; i<n; i++)
        fscanf(input,"%d\t%d\n",&x[i],&y[i]);

    fclose(input);
```

```
69
70          for(i=0;i<n;i++)
71          {
72              sumx=sumx +x[i];
73              sumx2=sumx2 +x[i]*x[i];
74              sumy=sumy +y[i];
75              sumxy=sumxy +x[i]*y[i];
76
77          }
78
79          c=((sumx2*sumy -sumx*sumxy)*1.0/(n*sumx2-sumx*sumx)*1.0);
80          m=((n*sumxy-sumx*sumy)*1.0/(n*sumx2-sumx*sumx)*1.0);
81          printf("\n\nThe line of best fit is y=%3.3fx +%3.3f \n\n",m,c);
82
83          return 0;
84      }
```

## 3. NUMERICAL INTEGRATION BY TRAPEZOIDAL RULE

```
85  #include<stdio.h>
86  #include<stdlib.h>
87  #include<math.h>
88
89  float functionOf(float x) {
90      // Replace the y = x^4 function as required below
91      return(pow(x,4));
92  }
93
94  int main() {
95      int i,n;
96      float x0,xn,h,t,y[20],so,se,ans,x[20];
97
98      system("clear");
99
100     printf("\n\nEnter the two limits:\t");
101     scanf("%f%f",&x0,&xn);
102     printf("\n\nEnter the width of an interval:\t");
103     scanf("%f",&h);
104
105     if(xn<x0){
106         t=xn;
107         xn=x0;
108         x0=t;
109     }
110
111     n=(xn-x0)/h;
112     if(n%2==1)
113     {
114         n=n+1;
115     }
116     h=(xn-x0)/n;
117
118     system("clear");
119
120     printf("\n\n\nSummary:\nUpper limit = %f\nLower limit = %f\nNumber
            of intervals = %d\nWidth of an interval ~ %f",xn,x0,n,h);
121     printf("\n\nThe values of y are\n");
```

```
122         for(i=0; i<=n; i++)
123         {
124             x[i]=x0+i*h;
125             y[i]=functionOf(x[i]);
126             printf("\t\t\ty%d = %f\n",i,y[i]);
127         }
128 
129         so=0;
130         se=0;
131 
132         for(i=1; i<n; i++)
133         {
134             if(i%2==1)
135             {
136                 so=so+y[i];
137             }
138             else
139             {
140                 se=se+y[i];
141             }
142         }
143         ans=h/3*(y[0]+y[n]+4*so+2*se);
144         printf("\nThe integration yields %f\n\n",ans);
145 
146         return 0;
147     }
```

## 4. NUMERICAL SOLUTION OF AN ORDINARY DIFFERENTIAL EQUATION BY RUNGE-KUTTA METHOD

```
148  #include<stdio.h>
149  #include<stdlib.h>
150  #include<math.h>
151
152  float functionOf(float x, float y){ // Replace function as required
         below
153      return (y+x)/(y*x);
154  }
155
156  int main(){
157      float K, K1, K2, K3, K4;
158      float x0 , y0, x, y, i, interval;
159      int j, n;
160
161      system("clear");
162
163      printf("\n\nSuggest any initial value for x:\t");
164      scanf("%f", &x0);
165      printf("\n\nSuggest any initial value for y:\t");
166      scanf("%f", &y0);
167      printf("\n\nEnter the number of iterations needed:\t");
168      scanf("%d", &n);
169      printf("\n\nEnter the skip between iterations:\t");
170      scanf("%f", &interval);
171      printf("\n\n");
172
173      x = x0;
174      y = y0;
175      for(i = x+interval, j = 0; j < n; i += interval, j++){
176          K1 = interval * functionOf(x , y);
177          K2 = interval * functionOf(x+interval/2, y+K1/2);
178          K3 = interval * functionOf(x+interval/2, y+K2/2);
179          K4 = interval * functionOf(x+interval, y+K3);
180          K = (K1 + 2*K2 + 2*K3 + K4)/6;
181          x = i;
182          y = y + K;
183          printf("\t\t\t x = %.2f \t y = %.4f\n", x, y);
```

```
184
185     // inf means infinity, NaN is an error for which we provide an
            explanation
186             if( isnan(x) || isnan(y) )
187                 printf("\nThere seems to be an error in line %d. \nCheck
                        your suggested initial values \nsince not all initial
                        values work \nfor all functions.",j);
188         }
189
190         printf("\n\n");
191         return 0;
192     }
```

## 5. PROJECTILE MOTION, INCLUDING STRING AND FILE HANDLING WITH AUTOMATIC GNUPLOT OUTPUT

```
193  #include<stdio.h>
194  #include<stdlib.h>
195  #include<math.h>
196  #include<unistd.h>
197  #include<string.h>
198
199  FILE *output;
200  float deg, v, g=9.81, d, x[100], y[100];
201  float rad, t1, t2, max, f=0.75;
202  float s[100],h[100];
203  int i=0, j=0, k=0, l=0, m=0, n=0, p=0;
204  int Y, N;
205  char filename[100], opt;
206  char * commandsForGnuplot[] = {"plot 'plot.temp' w linespoints"};
207
208  float angle(float deg){
209      rad=deg*M_PI/180;
210      return(rad);
211  }
212
213  float distance(float v, float rad){
214      d=v*v*sin(2*rad)/g;
215      return(d);
216  }
217
218  float height(float v, float rad){
219      max=v*v*sin(rad)*sin(rad)/(2*g);
220      return(max);
221  }
222
223  int main() {
224
225      system("clear");
226
227      printf("\n\nEnter the angle of throw:\t");
228      scanf("%f",&deg);
229      printf("\n\nEnter the velocity of throw:\t");
```

```
230         scanf("%f",&v);
231     ASK:printf("\n\nPlease pick a filename to output data into:\t");
232         scanf("%s",filename);
233         if(0 == access(filename, 0)){
234            printf("\n\nThis file exists. Do you want to add data to it (Y)
                  or create a new file (N)?\t");
235     CHOICE:scanf(" %c",&opt); // The space before %c here is extremely
           important.
236             Y=strncmp(&opt,"Y",1);
237             N=strncmp(&opt,"N",1);
238             if(Y==0){
239
240             }
241             else if(N==0){
242                 goto ASK;
243             }
244             else {
245                 printf("\n\nUnrecognised command. Please choose Y or N:\t");
246                 goto CHOICE;
247             }
248         }
249
250         FILE * temp = fopen("plot.temp", "w");
251         output=fopen(filename,"a+");
252
253         system("clear");
254
255         rad=angle(deg);
256         d=distance(v,rad);
257         max=height(v,rad);
258         t1=tan(rad);
259         t2=g/(2*v*v*cos(rad)*cos(rad));
260
261         j=0;
262         while(x[k]<=d){
263             x[k]=j*d/100;
264             y[k]=x[k]*t1-x[k]*x[k]*t2;
265             if(y<0)
266                 break;
267             fprintf(output, "%10.6f\t%15.6f\n",x[k],y[k]);
268             fprintf(temp, "%10.6f\t%15.6f\n",x[k],y[k]);
```

```
269             j++;
270         }
271         l=j;
272 
273         printf("\n\nAngle of throw in radian = %0.5f",rad);
274         printf("\n\nMaximum height = %0.3f",max);
275         printf("\n\nRange of the projectile = %0.3f",d);
276 
277         fclose(output);
278         printf("\n\nThe output has been written to %s.\n\n",filename);
279 
280         FILE * gnuplotPipe = popen ("gnuplot -persistent", "w");
281         fprintf(gnuplotPipe, "%s \n", commandsForGnuplot[i]);
282 
283         return 0;
284     }
```

## 6. GENERATE PASCAL'S TRIANGLE

**SPECIAL INSTRUCTIONS**

There are multiple versions of output possible for this program. Some output only half-a-triangle (right-angled) while others, such as this solution, output a full and proper triangle. However, due to spacing constraints the structure of the triangle breaks down if you attempt to output 17 or more rows. Furthermore, although the structure is intact, there is still a lack of spacing and the triangle becomes illegible if you attempt to output 14 or more rows.

```
#include<stdio.h>

int main() {

    int rows, coef=1, space, i, j;

    printf("Enter number of rows: ");
    scanf("%d", &rows);

    for (i=0; i<rows; i++) {
        for (space=1; space <= rows-i; space++)
            printf("  ");
        for (j=0; j<=i; j++) {
            if (j==0 || i==0)
                coef = 1;
            else
                coef=coef*(i-j+1)/j;
            printf("%4d", coef);
        }
        printf("\n");
    }

    return 0;
}
```

## 7. MULTIPLYING TWO MXN MATRICES

**NOTE**

This program can be slightly misleading. You might get an output without any errors but that does not mean your output is correct. Always multiply your input matrices by hand and use that result to cross-check your output. If you find it is incorrect, you might want to check line **??**.

```
309  #include<stdio.h>
310
311  int main() {
312
313      int matrixA[10][10], matrixB[10][10], matrixC[10][10] = {0};
314      int rowsA, rowsB, colsA, colsB, i, j, k;
315
316      // First, ask for number of rows and columns in the matrices
317      printf("\n\nEnter the number of rows and columns in matrix A:\n");
318      scanf("%d%d", &rowsA, &colsA);
319      printf("\n\nEnter the number of rows and columns in matrix B:\n");
320      scanf("%d%d", &rowsB, &colsB);
321
322      // Handle error if columns in matrix A is not equal to rows in B
323      while(colsA != rowsB) {
324          printf("\n\nMatrix B (%dx%d) is incompatible with matrix A (%dx%
                d).\nPlease re-enter the number of rows and columns in
                matrix B:\n", rowsB, colsB, rowsA, colsA);
325          scanf("%d%d", &rowsB, &colsB);
326      }
327
328      // Second, ask for elements of the matrices
329      printf("\n\nEnter elements of matrix A as asked:\n");
330      for(i = 0; i < rowsA; i++) {
331          for(j = 0; j < colsA; j++) {
332              printf("Enter element A%d%d:\t", i, j);
333              scanf("%d", &matrixA[i][j]);
334          }
335      }
336      printf("\n\nEnter elements of matrix B as asked:\n");
337      for(i = 0; i < rowsB; i++) {
338          for(j = 0; j < colsB; j++) {
339              printf("Enter element B%d%d:\t", i, j);
```

```
            scanf("%d", &matrixB[i][j]);
        }
    }

    // Multiply the two matrices
    for(i = 0; i < rowsA; i++) {
        for(j = 0; j < colsB; j++) {
            for(k = 0; k < colsA; k++) {
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }

    // Output the result
    printf("\n\nThe product is:\n");
    for(int i = 0; i < rowsA; i++) {
        for(int j = 0; j < colsB; j++) {
            printf("%d\t", matrixC[i][j]);
            if(j == colsB - 1) {
                printf("\n");
            }
        }
    }

    return 0;
}
```

## 8. FINDING THE SUM AND AVERAGE OF DATA STORED IN A FILE

**SPECIAL INSTRUCTIONS**

*Before starting this program, create a new folder and follow these instructions:*

1. Create a new text file using `vi data.txt` or equivalent.
2. Enter a column of numbers of your choice i.e. enter one number per line.
3. Save and quit with `:wq` and, optionally, use `ls` to ensure the file exists.
4. Type and execute your C program in the same folder as usual.

```
366  #include<stdio.h>
367
368  int main()
369  {
370      FILE *input;
371      int n,i;
372      float x[20],sum=0,avg;
373      char name[20];
374
375      printf("\n\nEnter the name of the input file with extension: \t");
376      scanf("%s", name);
377
378      printf("\n\nEnter the number of rows of data:\t");
379      scanf("%d",&n);
380
381      input = fopen(name,"r");
382
383      for(i=0; i<n; i++){
384          fscanf(input,"%f",&x[i]);
385          sum=sum +x[i];
386      }
387
388      fclose(input);
389
390      printf("\n\nThe sum is %3.2f.",sum);
391      printf("\n\nThe average is %3.2f.\n\n",sum/i);
392
393      return 0;
394  }
```