IC 150 P Computation for engineers lab

Lab assignment sheet no: 2, Odd semester, 2016

Basic C programming

Prepared by: Maben Rabi

In this lab session, you will learn and carry out some basic calculations in C. We will also make you use some slightly advanced tools from C programming. That is all right. In this sheet, we will tell you exactly what these tools are, and guide you on using them for the tasks of the session.

Programming is a lot like other creative activities like writing, drawing, or sculpture. The way to attack every programming task is to:

Plan Try to first digest the task you are attacking. If you would never use a computer to solve the task, how would you do it ? The answer to this question is the first step you take towards designing an *algorithm.* What are the variables that you take an inputs ? What are the variables that you will submit at part of the finished answer ? How can you arrive at the answer from the input variables ? Then make a flow chart out of your algorithm, and slowly change it so that every part of your modified flow chart can be given to a computer to compute. **Test your algorithm on a few test inputs.** For example, if you have finished designing an algorithm to check if a given positive integer is prime or not, then test this algorithm by trying out the first 10 natural numbers on it.

Make a rough sketch Now convert the algorithm you have into C code. Check if the C code faithfully computes what the algorithm says it should.

Get the details right Now check if the C program is a correct C program that compiles without error. Test the program on different reasonable inputs. Test it, test it, and keep testing it until you are satisfied that your program does what it is supposed to.

Debug To get the details right, you often have to debug. Debugging is for programmers what laborious experiments are for Chemists. This means that you should be prepared to carry out debugging with energy and cunning. If you do debugging like you are watching a time-pass movie, then you will be disappointed with the results. Passive debugging never works. You have to actively involve yourself. You have to look for clues; you have to divide your program into segments that you can test and check. Slowly and *systematically,* you check and make bigger and bigger portions of your code correct. And finally, you have the finished product in your hands !

You need to prepare your programs **before** you walk into the lab session. In fact you need to have your programs hand-written in your notebook. The tasks in this session will require on average **6 hour** of preparation before your arrive at the lab session. The PC lab is open for your use until late in the nights, and also in the weekends. Happy programming !

Objectives for this lab session

1. Learn to create programs that can implement some simple numerical calculations that use basic arithmetic operations of the C language, by reading Sections 1.1, 1.2 from Kernighan and Ritchie. and,
2. Learn to use the `if{...}else if{...}else{...}` command in the C language, by imitation and reading of Sections 3.1, 3.2, 3.3 from Kernighan and Ritchie.

Task 1: Simple calculations in combination with the `if` statement

Please read the programme typed in the next page, and try to understand what it does. In your own free time (before you come to the lab session), type this program and save it as the file `quadraticRoots.c`

```
1   #include<stdio.h>
2   #include<math.h>
3
4   void main(){
5     float a,b,c;
6     float discriminant,root1,root2;
7
8     printf("Enter a, b and c of quadratic equation:\n");
9     printf("\t\t a * x^2 + b * x + c = 0\n");
10    scanf("%f%f%f",&a,&b,&c);
11
12    discriminant = b * b - 4 * a * c;
13
14    if (a==0 && b==0 && c==0){
15      printf("Any real or complex number is a root.\n");
16    }
17    else if (a==0 && b==0 && c!=0){
18      printf("No root exists.\n");
19    }
20    else if(discriminant < 0){
21      printf("Roots are complex number. Roots are");
22
23      printf("%.3f%+.3fi",-b/(2*a),sqrt(-discriminant)/(2*a));
24      printf(", %.3f%+.3fi\n",-b/(2*a),-sqrt(-discriminant)/(2*a));
25    }
26    else if(discriminant==0){
27     printf("Both roots are equal. Root is: ");
28
29     root1 = -b /(2* a);
30     printf(" %.3f\n ",root1);
31    }
32    else{
33     printf("Roots are real numbers. Roots are: ");
34
35     root1 = ( -b + sqrt(discriminant)) / (2* a);
36     root2 = ( -b - sqrt(discriminant)) / (2* a);
37     printf("%.3f , %.3f\n",root1,root2);
38    }
39  }
```

quadraticRoots.c

Compile it using the command (from inside the same directory where the file is):
`gcc quadraticRoots.c -o quadraticRoots -lm`

Do not worry about how strange this command seems. In a few weeks you will understand what it means. To execute the compiled program, type (in the same directory):
`./quadraticRoots`
Play with the program. Try to understand what it does. While reading the program for the first three times, please ignore the exact details of the `printf, scanf` commands. The details will look strange now, but you will get used to them in a few weeks.

The program `quadraticRoots.c` has 39 lines of code. If you find it intimidating, then make the following shorter program `quadraticRootsSkeleton.c` out of lines from the original program `quadraticRoots.c`

```
1   #include<stdio.h>
2   #include<math.h>
3
4   void main(){
5     float a,b,c;
6     float discriminant,root1,root2;
7
8     printf("Enter a, b and c of quadratic equation:\n");
9     printf("\t\t a * x^2 + b * x + c = 0\n");
10    scanf("%f%f%f",&a,&b,&c);
11
12    discriminant = b * b - 4 * a * c;
13     root1 = ( -b + sqrt(discriminant)) / (2* a);
14     root2 = ( -b - sqrt(discriminant)) / (2* a);
15     printf("%.3f , %.3f\n",root1,root2);
16  }
```

selected lines from `quadraticRoots.c`

Play with this shorter program, and try to understand what it does. Be warned that the shorter program above will work properly only for some combinations of the inputs. Then, carefully consider what lines to add from the original file, and try to make it work.

Your programming task

By imitating the program given above, write a C program that will compute the root of the equation

$$ax + b = 0.$$

Task 2: Calculations with regular polygons

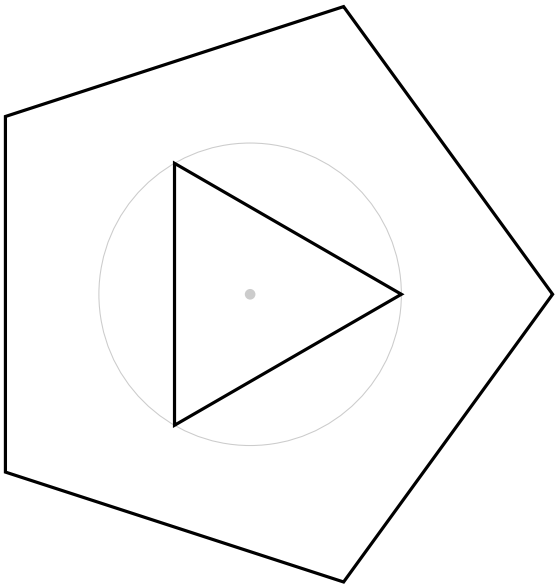
Write a C program that will take in the number of sides and the side length of a regular polygon, and print out the vertex angle of the regular polygon, and its area. (hint: The number of sides will be stored as an integer, and the side length will be stored as floating point number).

Task 3: Will one fit into the other ?

We say regular polygon A freely fits into regular polygon B if the following two conditions hold:

- The centres of both polygons coincide
- When the common centre is fixed, regular polygon A must lie completely inside regular polygon B no matter how A and B are rotated with respect to each other. The picture below illustrates two cases, one with a fit and another without.

one fits into the other



neither fits into the other

Write a C program that will take in details about two regular polygons and print out whether or not one regular polygon will freely fit into the other. The program must take in the number of sides and the side length of a regular polygon A. Then it must take in the number of sides and the side length of a regular polygon B. Then it must print whether or not one regular polygon fits into the other.

References

The C programming language, Kernighan and Ritchie, second edition, Prentice Hall of India.