Modeling a Robotic Arm in C++: Variables, Data Types, and Assignment Operation

In computational science and engineering, there are five identifiable stages in the computer solution of a problem. The first stage consist in presenting a physical problem. A mathematical model derived for this problem is the second stage. An appropriate solution method is next. The following stage is the implementation in a C++ program. The final stage is the assessment.

In this lesson we use the modeling of a robotic arm to present the stages of computational science and engineering. The case study considers the kinematics of a robotic arm. We use this opportunity to discuss several ideas about programming in the C++ language. We will discuss how variables, data types, and assignments are expressed in C++. Each of these stages is presented as a separate section is this lesson.

1.1 Problem Description

The robotic arm consist of two limb sections, the arm and the forearm, connected at a joint, the elbow. The arm is attached at the other end to the body, and the forearm could connected to a hand. The limbs are connected at their endpoints by joints, shoulder and elbow, that are allowed to rotate in a planar motion.

The problem is to determine the X-Y position of the robotic wrist joint given the angle arm makes with the body; the angle that the forearm makes with the arm, and the length of the limb segments.
1. Modeling a Robotic Arm in C++: Variables, Data Types, and Assignment Operation

1.2 A Model

We can use a link diagram to describe a model of the motion of the robotic arm. The Figure below represents the arm and forearm as links joined at the elbow. We will consider the two dimensional motion represented by the Figure.

![Figure 1.1: Robotic Arm](image)

The X-Y position of the shoulder joint is fixed. The top endpoint of the arm link is attached at the shoulder joint. The arm can rotate around the shoulder joint. The arm must always be attached to the shoulder. The forearm link is connected to the arm link, and the hand link would be connected to the arm link.

We represent the movement of the robotic arm by the coordinates of the joints in the X-Y plane. The motion of the arm is determined by the angle, alpha, that the arm makes with the body, by the angle, beta, that the forearm makes with the arm, and by the lengths of the links. The lengths of the segments are shown in the link diagram.

The angle alpha will be measured in degrees counterclockwise from the positive Y axis. The angle beta will be measured in degrees clockwise from the arm to the forearm. When the arm is raised to the horizontal position, the angle alpha is degrees and the angle beta is 180 degrees.
1.3 The Solution Method

The problem is to determine the X-Y position of the robotic wrist joint given the angle arm makes with the body, the angle that the forearm makes with the arm, and the length of the link segments. Let us consider solving first for the X-Y position of the robotic elbow joint.

A bit of trigonometry is all that is necessary to solve the problem analytically. From robot Figure, the equations that describe the X-Y position of the robotic elbow joint as functions of the shoulder angle alpha, are:

\[ y_{\text{elbow}} = y_{\text{shoulder}} - l_{\text{arm}} \times \cos(\alpha) \]

and

\[ x_{\text{elbow}} = x_{\text{shoulder}} + l_{\text{arm}} \times \sin(\alpha) \]

We have solved robotic elbow joint X-Y position analytically. Let us now solve the problem with a C++ program.

1.4 Implementation in C++

The C++ program rbtjnts.cpp implements the analytical solution. You should be able to look at rbtjnts.cpp and have a pretty good idea how the program works. We will talk about the individual statement in the program in just a moment.

```cpp
// This is the file "rbtjnts.cpp".
#include <iostream>
#include <cmath>
using std::cout;
using std::cin;
using std::endl;

int main()
{

    /*
     * Written by JCDiaz June 30 1999.
     * Modified Jan 16 2004
     *
     * This program determines the X-Y position of a robotic elbow joint.
     *
     * Input:
     *  Predetermined.
     *     -- The X-Y position of the shoulder joint
     *     -- The length of the arm.
     *  The user is prompted by the program to enter:
     *     -- The angle of the shoulder joint in degrees
     *           counterclockwise from the positive X axis.
     *
     * Output:
     *  The output is to the screen:
     *     -- The X-Y position of a robotic elbow joint
     */

    //...
shoulderx and shouldery are the X and Y coordinates of the shoulder joint.
arm is the length of the robotic arm, in meters.

alphad is the angle of the shoulder joint, measured in degrees.
alphar is the angle of the shoulder joint, measured in radians.

eльbowx and elbowy are the x and y coordinates of the elbow joint.
*/

// no undefined variables

float shoulderx, shouldery;
float arml;
float alphad, alphar;
float elbowx, elbowy;

const float pi = 3.14159;  // Declares pi to be a constant.

// Set up the known values.

shoulderx = 0.0;
shouldery = 1.45;
arml = 0.343;

// Request the angle of the shoulder joint from the user.

cout << "Enter the angle of the shoulder joint (in degrees): " << endl;
cin >> alphad;
alphar = alphad * (pi / 180.0); // Convert to radians.

// Determine the (x, y) position of the elbow joint.

e1bowx = shoulderx + (arml * sin(alphar));
elbowy = shouldery - (arml * cos(alphar));

// Print out the results.

cout << "The X coordinate of the elbow joint is " << elbowx << endl;
cout << "The Y coordinate of the elbow joint is " << elbowy << endl;

} // End of file.

Take a moment to read the program and try to understand how it determines the position of the robotic
elbow joint. Don’t worry if you don’t “know” how to write C++. Using your own knowledge of the
problem, you should be able to figure out, more or less, what the programs do and how they do it.

1.4.1 Program Structure

Reading and having examined, compiled, and run the sample C++ program, gives an idea what it does
and how it works. The specific syntax rules for writing C++ programs are yet to be discussed.
Before we get into a line-by-line discussion, however, let's take a closer look at the overall structures of the program rbtjnts.cpp. Note the generous use of comments. There is a preamble that describes the object of the program. The variable description section describes the named variables involved in the program. This is followed by the actual variable declaration. After the variable declarations, the actual program statements follows. We will discuss the individual statements in the program in this lesson. The last program section before the ending } is the output section.

Variables in C++

The concept of a variable needs to be introduced first. A variable is a storage location with a name. You can store a value in a variable. You can later examine, use, or change that value. C++ uses variables to keep track of its computations. But each variable that is going to be used in a C++ program must be declared first. In other words, before you use a variable in a C++ program, you must tell the compiler something about that variable. You must tell the compiler two things about every variable: the name of the variable and the type of the variable. The compiler requires that every variable has a declaration.

C++ Variable Names

As you already know, every variable has a name. C++ has rules about the names that you can give to variables. The variable declaration float shoulderx, shouldery; specifies that the variables are of float type and that the names are shoulderx, shouldery. The name of a variable can be any string formed with any letter, digits, or the underscore sign. The first character in a variable name can not be a digit. The length name of the variable is restricted in other languages but not in C++.

- X is an acceptable C++ variable name.
- EPSILON is acceptable.
- abc123 is acceptable.
- 123ABC is not acceptable, it starts with a digit.
- 1/mu is not acceptable, it not only starts with a digit but it also uses the symbol / which is reserved in C++.

What about these names: alpha, X+Y, HuNdReDtH, Result, and x12?

C++ Type Declarations

In C++ each variable is defined to hold exactly one kind of data. For example, a variable named X might be able to hold integers, or it might be able to hold real numbers, but it can’t hold both kinds of numbers. The Type of a variable is the kind of data it can hold.

Every variable in a C++ program has a specific type. The most useful types in C++ are:
C++ type | Typical size (bytes)
---|---
char | 1
int | 4
float | 4
double | 8

Right now, the types that are of most interest to us are the numerical types. A C++ variable is declared by including a statement containing the variable’s type and name on a program before it is used on the right side of another program statement. For example, the statement:

```cpp
int count;
```

declares that there is a variable named `count` and that it can contain integer values (and only integer values). You can declare several variables of the same type on a single statement by separating the variable names with commas. For example:

```cpp
float shoulderx, shoulderY;
```

declares that there are two variables named `shoulderx` and `shouldery` that can contain real values. Look at the `rbtjnts.cpp` program and identify the variable declarations.

Additional data type qualifiers for integers are also possible in C++. The reserved words `long`, `short` and `unsigned` can be combined with `int` to define other type of integer variables

```cpp
long int large_number; // for integers known to be very large.
short int low_number; // for integers known to be low.
unsigned int positive_number; // for integers known to be positive.
```

It is also possible to declare `long double`.

**Assignment Operation**

A very important kind of C++ program construct is the **assignment operation**. Once a variable has been declared, it can be given a value through an assignment operation. In C++ the assignment operator is the `=` sign. For example, the C++ statement:

```cpp
shoulderx = 0.0;
```

assigns the (floating point) value zero to the (floating point) variable `shoulderx`. It is important to remember that assignment is not the same thing as equality or definition. For example, consider the following C++ statement:

```cpp
count = count + 1;
```

As an algebraic equation, that statement is nonsense. If you tried to solve it for the value of `count`, there would be no solution to the equation. But assignment statements are not algebraic equations. The above statement tells the computer to determine the current value of the variable `count`, add one to that value, and then store that result back in the variable `count`.

The C++ assignment operation has the form:

```cpp
variable_name = expression ;
```
where variable_name must be an already declared C++ variable, and expression is a valid C++ expression which produces a result. The meaning of a C++ assignment statement is to store the resulting value of calculating the right hand expression into the variable named on the left hand side. Typically the expressions encountered in this class are arithmetic expressions resulting in arithmetic values. Pause now and locate all of the assignment statements in the program rbtjnts.cpp.

Technically the assignment operator takes two operands, the left operand must be a variable name, and the right operand must be an expression resulting in a similar object value as the named variable. If the types are not consistent, like attempting to store a numeric result into a character variable, the compiler will identify it as an error.

An expression may never appear in the left side of a C++ assignment. The PVT law for ideal gases state that

\[ PV = nRT \]

where \( P \) is the pressure, \( V \) is the volume, \( T \) is the temperature, \( R \) is the gas constant, and \( n = N/N_A \), with \( N_A \) Avogadro's number and \( N \) is the number of particles in a box of volume \( V \). The ideal gas laws is an equation that can be used to solve for any of its component variables in terms of the others. In order to obtain the Temperature given the pressure, volume and \( N \), it is necessary to solve first for \( T \) form the equation:

\[ T = \frac{PV}{nR}, \]

before rendering it into C++.

The algebra must precede the program statements! For instance, consider the mathematical expression:

\[ \frac{1}{x} = 10.2 + 5.2(p + 8079 + (p - 3.4)^{2.5}). \]

It must be rewritten as:

\[ x = \frac{1}{10.2 + 5.2(p + 8079 + (p - 3.4)^{2.5})} \]

before rendering it in C++ as shown in the following section of code.

```c++
float x, p;
p = ... ; /* p is assigned some value */
x = 1/ (10.2 + 5.2 * ( p + 8079 + pow(p-3.4,2.5)));
```

### Arithmetic Operations

C++ supports the basic arithmetic operators:

- `+` for addition,
- `-` for subtraction,
- `*` for multiplication,
- `/` for division.

Significantly, C++ does not support exponentiation as an operator, one must invoke the `pow()` function, see description below.

You can group together parts of an expression with parentheses.
result = (a + b) * c;

Our example C++ program contains examples of the basic arithmetic operators.

Common C++ Functions

C++ provides for built-in procedures for certain mathematical functions. To invoke a built-in function, you write the name of the function followed by a list of arguments in parentheses. To use the sine function in C++ statement, you would write:

\[ ... = ... \sin(\ angle) ... \]

where the \textit{angle} is replaced with an expression that is the angle you want to take the sine of. The angle is measured in radians. The description of the \texttt{sin()} and \texttt{cos()} functions are included in the built-in header file \texttt{cmath} which has been included at the top of the file. This is a \texttt{preprocessor statement} that directs the compiler to look for the definition of the function in the header file. Some compilers require the additional flag -lm to indicate that the math library must also be loaded.

There are other built-in mathematical functions in the \texttt{cmath} collection. Below partial list of the built-in mathematical functions is presented. The robot problem only needed sine and cosine. Future lessons will present how are user defined functions built.

- \texttt{fabs(x)}: absolute value of \(x\), returns same type as input.
- \texttt{pow(y,p)}: \(y\) raised to the \(p\) power, returns the same type as \(y\).
- \texttt{sqrt(z)}: square root of \(z\), returns the same type as input.
- \texttt{tan(theta)}: tangent of \(\theta\) (radians are expected), returns a real value.
- \texttt{asin(z)}: arc sine of \(z\), \(-1 \leq z \leq 1\) is expected, output value is in radians, returns a real value.
- \texttt{acos(z)}: arc cosine of \(z\), \(-1 \leq z \leq 1\) is expected, output value is in radians, returns a real value.
- \texttt{atan(z)}: arc tangent of \(z\), output value is in radians, returns a real value.
- \texttt{log(x)}: natural logarithm of \(x\), \(x \neq 0\) is expected, returns a real value.
- \texttt{log10(x)}: log base 10 of \(x\), \(x \neq 0\) is expected, returns a real value.
- \texttt{exp(k)}: \(e\) raised to power \(k\), returns a real value.
- \texttt{copysign(a,b)}: the sign of \(b\) times the magnitude of \(a\), returns a real value.

Input and Output

Most programs need to communicate with something outside themselves in order to be useful. The “outside world” can be a human being, or a data file, or a printer. The process of retrieving data from that outside world is called \texttt{input}, and the process of sending messages to the outside world is called \texttt{output}. These two concepts are often grouped together and abbreviated as I/O which stands for \texttt{input/output}. In a previous lesson file I/O was discussed.

Communicating with the outside world can be a very complicated task for a program. It can require a lot of programmer effort to get the output from a program to look just right or to handle all of the various problems that might arise. For example, a program that sends output to a printer might want to do something special when the printer runs out of paper.
Program Structure

In summary in a program variables must be defined, variables are initialized through known variables, or input from either a file or from the user, arithmetic expressions which may involve functions of these variables are calculated, their results stored in variables through assignment statements, and final and intermediate results are output for users to view.

A C++ program has the following structure:

```cpp
/* Begin program file */
/* Begin preprocessor and header file section */
#include <iostream> /* suitable I/O file stream */
#include <cmath> /* when necessary for invocation of certain Mathematical functions */
/* End preprocessor and header file section */

/* Program preamble which describes the object of the program its required input and its output */

int main()
{
    /* Variable declaration section */
    /* Variable initialization and input section */
    /* Statements carrying out the calculations */
    /* Output section */
}
/* End of Program file */
```

Coulomb's Law

Coulomb's law relates the energy of interaction between a pair of ions and the distance between their centers and their numerical ion charges. The energy interaction in Jules $E$ between a pair of ions is given by

$$E = 2.31 \times 10^{-19} \frac{Q_1 Q_2}{r}$$

Where $r$ is the distance between the ion centers in nm, and $Q_1$ and $Q_2$ are the numerical ion charges.

A simple main() program that receives as input the distance between the ion centers and the numerical ion charges, and outputs the energy interaction is presented below. The input is from the file `Coulomb.txt`. The results are sent to the file `EnergyInter.txt`.

```cpp
/* Begin program file */
/* Begin preprocessor and header file section */
#include <fstream> /* suitable I/O file stream */
#include <cmath> /* when necessary for invocation of certain Mathematical functions */
```
1. Modeling a Robotic Arm in C++: Variables, Data Types, and Assignment Operation

/* End preprocessor and header file section */

using std::ifstream;
using std::ofstream;
using std::endl;

ifstream fin("Coulomb.txt");
ofstream fout("EnergyInter.txt");

/* Program preamble which describes the object of the program its required input and its output */
/* Coulomb's law relates the energy of interaction between a pair of ions and the distance between their centers and their numerical ion charges. the program that receives as input the distance between the ion centers and the numerical ion charges, and outputs the energy interaction. *----------------------------------------------------------------
The variables are : Q1, Q2 are the numerical ion charges. r is the distance between the ion centers in nm. EnergInter is the energy interaction. */

int main()
{
    /* Variable declaration section */
    /* All C++ variables must be declared before use. */
    float Q1, Q2, r, EnergInter;

    /* Variable initialization and input section */
    fin >> Q1 >> Q2;
    /* Statements carrying out the calculations */
    EnergInter = 2.31E-9 * Q1*Q2 / r;
    /* Output section */
    fout << "The Energy Interaction is " << EnergInter << endl;
}
/* End of Program file */

1.5 Assessment

1.5.1 Exercises

In the following a few exercises of simple physical and chemical laws are provided.
Nernst Equation

Nernst equation describes the relationship between the cell potential and the concentration of the cell components. The cell potential $E$ is given by:

$$E = E^o - \frac{RT}{nF} \ln(Q).$$

Where $E^o$ is the cell potential at standard conditions, $n$ is the number of moles of electrons, $F$ is the charge per mole of electrons, $T$ is the temperature, $R$ is the gas constant, and $Q$ is the reaction quotient.

Write a main() program that receives as input the cell potential at standard conditions, the number of moles of electrons, the charge per mole of electrons, the temperature, and the reaction quotient; and outputs the cell potential. The input is from the standard input, the keyboard. The results are sent to the standard output, the screen.

Bingham’s correlation Lab

Bingham’s correlation for the viscosity of water as a function of temperature is given by:

$$\frac{1}{\mu} = 0.021482 \ast (T - 8.435 + \sqrt{(8078.9 + (T - 8.435)^2)}) - 1.2$$

where $\mu$ is the viscosity in centipoise and $T$ is the temperature in oC.

1. Create a directory called cs2503Programs in your utulsa file server. Make sure it is accessible only by you. The Unix command chmod go-rx cs2503Programs would do it.

2. Write a main() program that receives as input the temperature and outputs the viscosity of water. The input is from the standard input, the keyboard. The results are sent to the standard output, the screen. Save it as BinghamWatVisc.cpp in your directory cs2503Programs.

   Please note: you are required to maintain the user restrictions to this directory. You are also expected to store ALL the programs related to this class in this directory. Any of these violations will be considered an academic infraction and severe penalties will apply including an F grade for the class. This will be continually monitored by the Instructor and the TA periodically.

3. Verify that you typed it correctly by using the Unix command: g++ BinghamWatVisc.cpp -o BinghamWatVisc. You should have now a new file named BinghamWatVisc. Otherwise, you have miss typed something and would need to correct it.

4. Once your program compiles, verify that it produces the correct results as follows:
   - Water Temperature 90oC, Bingham water viscosity is 1.70389cp
   - Water Temperature 100oC, Bingham water viscosity is 1.24703cp

5. Once your program works correctly, create a file called BinghamWatViscLabReport.html which explains what you accomplished in this lab and save it in your public_html/cs2503Reports directory. Make sure that your file BinghamWatVisc.cpp is not accessible by any other user.

6. Modify your cs2503.html file to place a link to BinghamWatViscLabReport.html.

7. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.

1.5.2 Extensions of the Robot’s Arm Program

We can extend the solution method to solve for the X and Y coordinate position of the robotic wrist joint. Once the position of the elbow joint is known, two more pieces of information are required to locate the wrist:

1. The length of the robotic forearm.

2. The angle of the robotic elbow joint.

The rbtjnts.cpp program can also be modified to compute the wrist position. First of all appropriate variable declarations need to be added to the program near the declarations already in place. The new variable declarations would probably look something like this:

```cpp
float forearml;
float elbowd, elbowr;
float wristx, wristy;
```

where:

- `forearml` is the length of the robotic forearm link,
- `elbowd` is the angle of the robotic elbow joint in degrees,
- `elbowr` is the angle of the robotic elbow joint in radians, and
- `wristx` and `wristy` are the X and Y coordinates of the wrist.

We must not forget to include comments in the program that describe these new variables.

1.5.3 Elbow and Wrist of a Robot Project

In this project we will practice adding new features to a C++ program so it can solve a more complex problem. Consider the robot arm. With the information in the link diagram, plus the values of the angles of the robot arm joints, it is possible to determine the X-Y positions of the robot arm joints.

1. Determine the equations for locating the elbow and wrist joints.
2. Copy the file `rbtjnts.cpp` into the file `robotEW.cpp` and save it in your directory `cs2503Programs`.
3. Change directory into your `cs2503Programs` directory.
4. Modify the `robotEW.cpp` program as follows:
   - The program shall ask the user for the current angles of the robotic arm shoulder, and elbow joints. The angles shall be input in degrees. The angles shall be input as real numbers.
1. **Modeling a Robotic Arm in C++: Variables, Data Types, and Assignment Operation**

- The program shall use the input angles and the values shown in the link diagram to compute the positions of the robotic arm elbow, and wrist joints.
- The program shall print out the X and Y coordinates of the robotic arm elbow, and wrist joints.

5. Modify the program `robotEW.cpp` so that it prints out to the screen the input it received. Make sure that when it prints, it identifies the values it is printing.

6. Modify the program so that it reads from a file named `robotEWin.txt`. Further modify your program so that all printing is done to a file named `robotEWresults.txt`.

7. Once you have saved the file `robotEW.cpp`, and it works correctly, handin the files `robotEW.cpp`, `robotEWin.txt`, and `robotEWresults.html`.

8. Copy the files `robotEWin.txt`, and `robotEWresults.txt` into your `public_html/cs2503Reports` directory.

9. Change directory to your `public_html/cs2503Reports` directory.

10. Write a file `robotArmEqs.html` that describes the equations required for computing the elbow and wrist joints. Provide a description of each parameter, and identify which are known and which are determined.

11. Create the report file `robotEWreport.html` which explains the project and includes relative links to `robotArmEqs.html`, `robotEWin.txt`, and `robotEWresults.txt`.

12. Modify your `cs2503.html` file to place a link to the report file.

13. Handin the report file `robotEWreport.html` and your modified `cs2503.html`.

14. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.