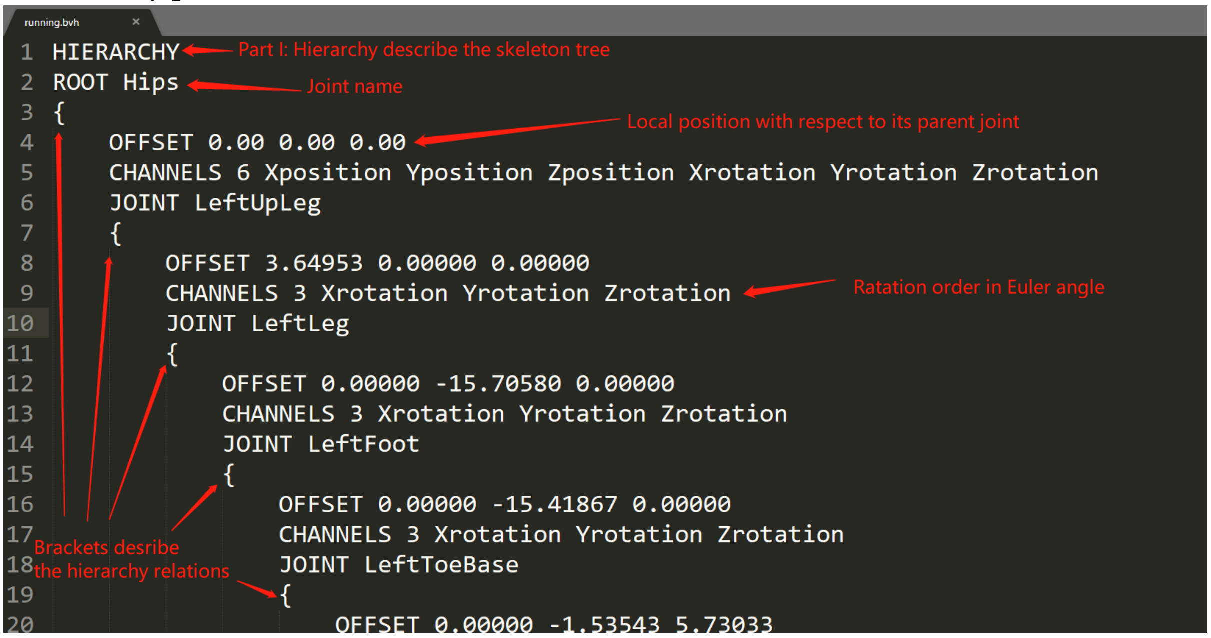
# CMPT466-888 Computer Animation

Programming Assignment 1: Forward Kinematics

In this lab, you are going to apply Forward Kinematic method with motion data obtained from a BVH file. The default coding environment is Visual Studio 2017 Community (workload: Desktop development with C++). Please ensure that your submission is runnable on the desktops with VS2017 (**x64 compiler**) and Windows 10 in ABS9840.

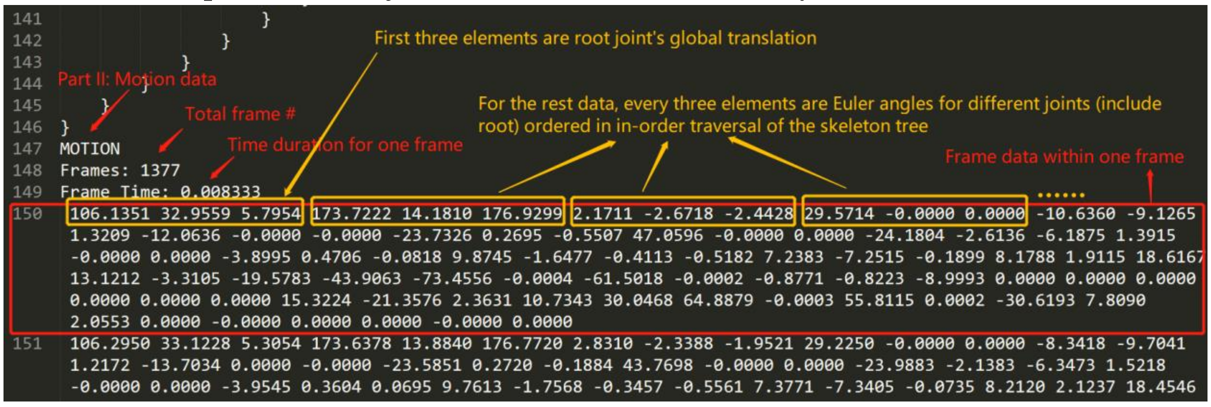
## Introduction to BVH file

A BVH file contains two parts:
1) Hierarchy part describe the skeleton tree.


running.bvh
Part I: Hierarchy describe the skeleton tree
Joint name
Local position with respect to its parent joint
Ratation order in Euler angle
Brackets desribe
the hierarchy relations

```
HIERARCHY
ROOT Hips
{
    OFFSET 0.00 0.00 0.00
    CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
    JOINT LeftUpLeg
    {
        OFFSET 3.64953 0.00000 0.00000
        CHANNELS 3 Xrotation Yrotation Zrotation
        JOINT LeftLeg
        {
            OFFSET 0.00000 -15.70580 0.00000
            CHANNELS 3 Xrotation Yrotation Zrotation
            JOINT LeftFoot
            {
                OFFSET 0.00000 -15.41867 0.00000
                CHANNELS 3 Xrotation Yrotation Zrotation
                JOINT LeftToeBase
                {
                    OFFSET 0.00000 -1.53543 5.73033
```

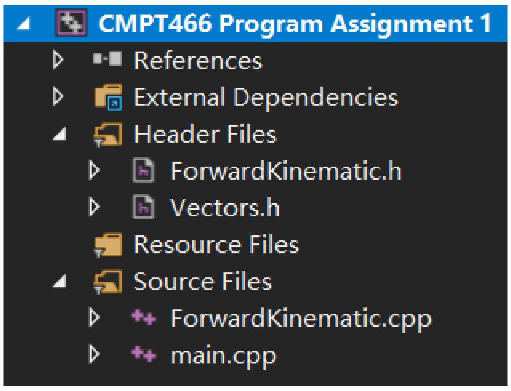
2) Motion data part stores all joints' transformations, frame by frame.


Part II: Motion data
Total frame #
Time duration for one frame
First three elements are root joint's global translation
For the rest data, every three elements are Euler angles for different joints (include root) ordered in in-order traversal of the skeleton tree
Frame data within one frame
......

```
                        }
                    }
                }
            }
        }
}
MOTION
Frames: 1377
Frame Time: 0.008333
106.1351 32.9559 5.7954 173.7222 14.1810 176.9299 2.1711 -2.6718 -2.4428 29.5714 -0.0000 0.0000 -10.6360 -9.1265
1.3209 -12.0636 -0.0000 -0.0000 -23.7326 0.2695 -0.5507 47.0596 -0.0000 0.0000 -24.1804 -2.6136 -6.1875 1.3915
-0.0000 0.0000 -3.8995 0.4706 -0.0818 9.8745 -1.6477 -0.4113 -0.5182 7.2383 -7.2515 -0.1899 8.1788 1.9115 18.6167
13.1212 -3.3105 -19.5783 -43.9063 -73.4556 -0.0004 -61.5018 -0.0002 -0.8771 -0.8223 -8.9993 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 15.3224 -21.3576 2.3631 10.7343 30.0468 64.8879 -0.0003 55.8115 0.0002 -30.6193 7.8090
2.0553 0.0000 -0.0000 0.0000 0.0000 -0.0000 0.0000
106.2950 33.1228 5.3054 173.6378 13.8840 176.7720 2.8310 -2.3388 -1.9521 29.2250 -0.0000 0.0000 -8.3418 -9.7041
1.2172 -13.7034 0.0000 -0.0000 -23.5851 0.2720 -0.1884 43.7698 -0.0000 0.0000 -23.9883 -2.1383 -6.3473 1.5218
-0.0000 0.0000 -3.9545 0.3604 0.0695 9.7613 -1.7568 -0.3457 -0.5561 7.3771 -7.3405 -0.0735 8.2120 2.1237 18.4546
```

## Where to code?

The template is given as `CMPT466-888 Programm Assignment 1.zip`. Extract it. You can play with the result demo by executing the .exe file in the folder `ResultDemo` first. Open `build/CMPT466_Program_Assignment_1.sln` with **Visual Studio 2017.**



```
◢ CMPT466 Program Assignment 1
  ▷ References
  ▷ External Dependencies
  ◢ Header Files
    ▷ ForwardKinematic.h
    ▷ Vectors.h
    Resource Files
  ◢ Source Files
    ▷ ForwardKinematic.cpp
    ▷ main.cpp
```

1) `Vectors.h` Do not edit this file.

   Support file contains many vector manipulation features (e.g., dot product). Feel free to use them in `main.cpp`: main entry, contains opengl initial operations.

2) `ForwardKinematic.h` Do not edit this file.

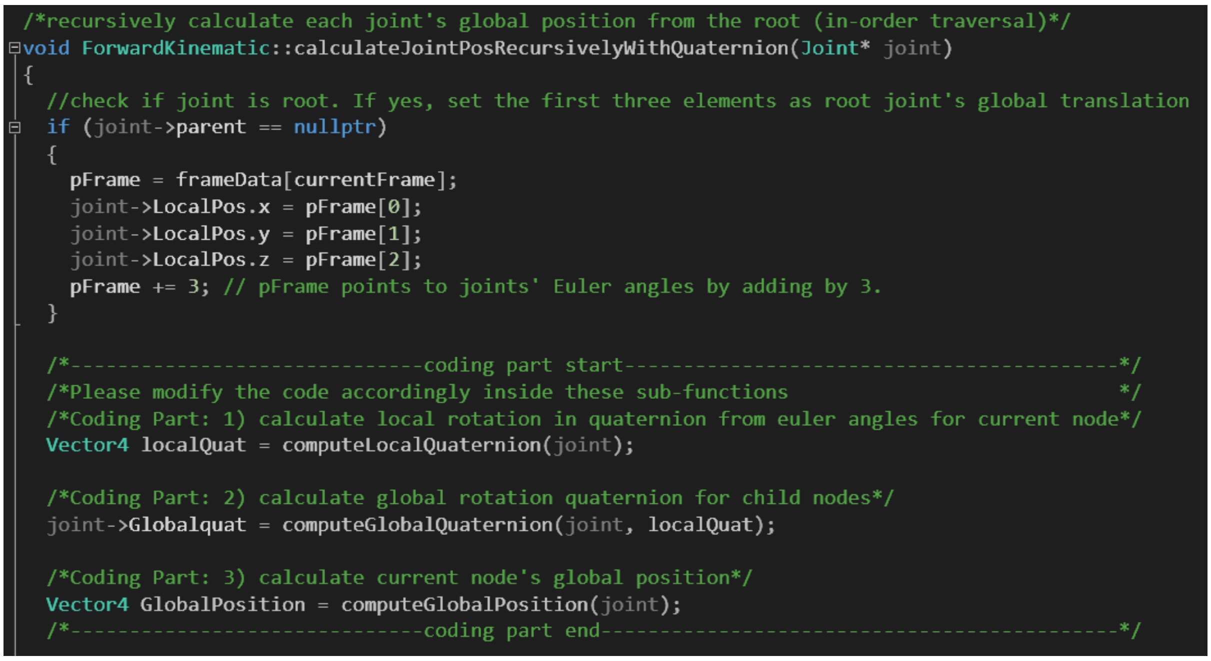
   Contains default declarations of functions and variables.

3) `ForwardKinematic.cpp` The only file you need to modify.

   Contains the implementation of functions in `ForwardKinematic.h`.

There are **three sub-functions** needed to be completed within the recursive function in `ForwardKinematic.cpp`.

```
calculateJointPosRecursivelyWithQuaternion(Joint* joint)
|-- computeLocalQuaternion(Joint* joint)
|-- computeGlobalQuaternion(Joint* joint, Vector4 localQuat)
|-- computeGlobalPosition(Joint* joint)
```



```
/*recursively calculate each joint's global position from the root (in-order traversal)*/
void ForwardKinematic::calculateJointPosRecursivelyWithQuaternion(Joint* joint)
{
  //check if joint is root. If yes, set the first three elements as root joint's global translation
  if (joint->parent == nullptr)
  {
    pFrame = frameData[currentFrame];
    joint->LocalPos.x = pFrame[0];
    joint->LocalPos.y = pFrame[1];
    joint->LocalPos.z = pFrame[2];
    pFrame += 3; // pFrame points to joints' Euler angles by adding by 3.
  }

  /*-----------------------------coding part start-----------------------------------------*/
  /*Please modify the code accordingly inside these sub-functions                         */
  /*Coding Part: 1) calculate local rotation in quaternion from euler angles for current node*/
  Vector4 localQuat = computeLocalQuaternion(joint);

  /*Coding Part: 2) calculate global rotation quaternion for child nodes*/
  joint->Globalquat = computeGlobalQuaternion(joint, localQuat);

  /*Coding Part: 3) calculate current node's global position*/
  Vector4 GlobalPosition = computeGlobalPosition(joint);
  /*-----------------------------coding part end-------------------------------------------*/
```

This function would access every joint in **in-order traverse** of the skeleton tree and calculates every joint's global position for one frame.

1) `FrameData` is a `float**` type pointer (2D array) to all the motion data within `running.bvh`.

2) `pFrame` is pointing to one row (one frame's motion data) of `frameData` indexed by `currentFrame`.

**You may need to:**

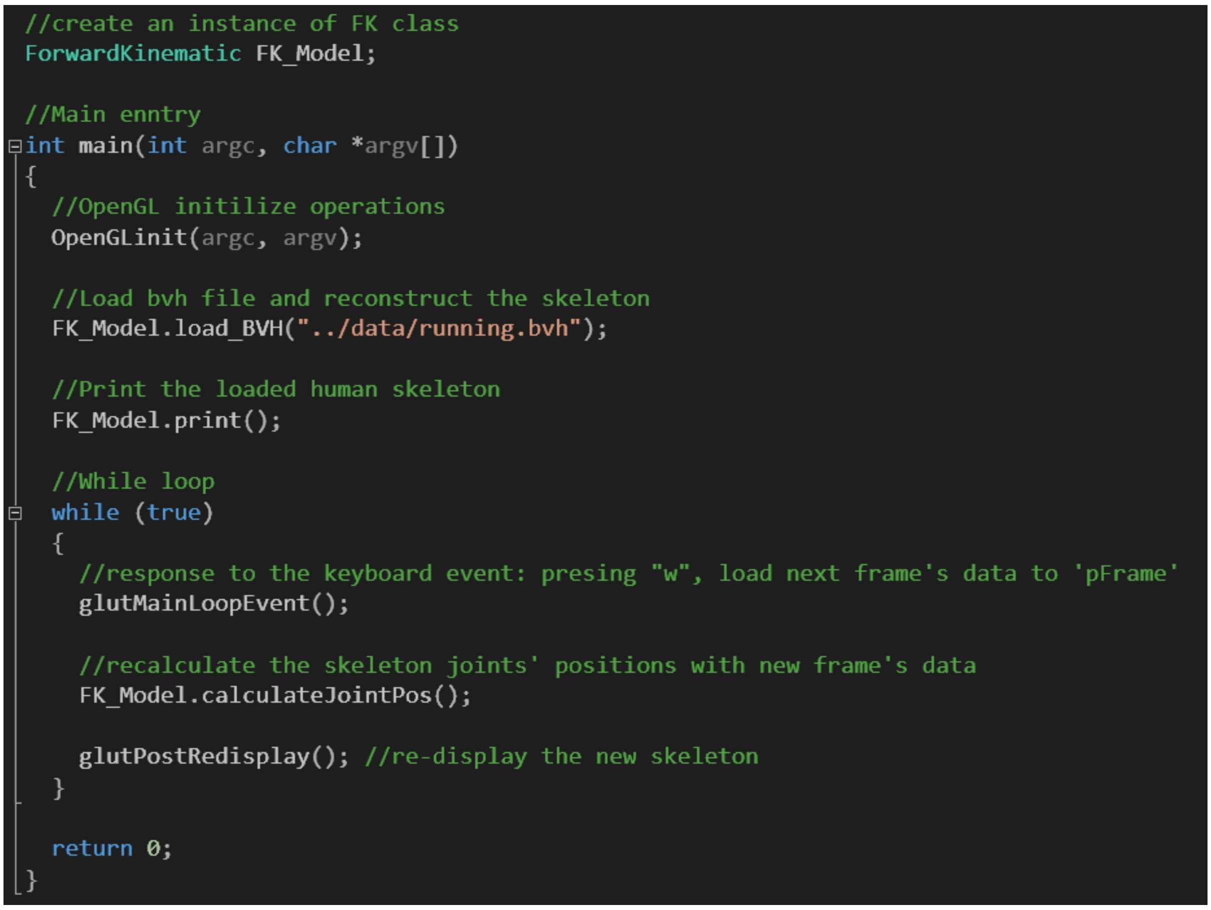
1) use `Vector4` class to represent quaternions. check more details in `Vectors.h`.

2) use `quaternionMultiplication(Vector4, Vector4)` to calculate quaternion multiplication

3) use the default skeleton tree's root joint (`root`) and `pFrame` to calculate every joint's global position (in quaternions).

## Introduction to the project

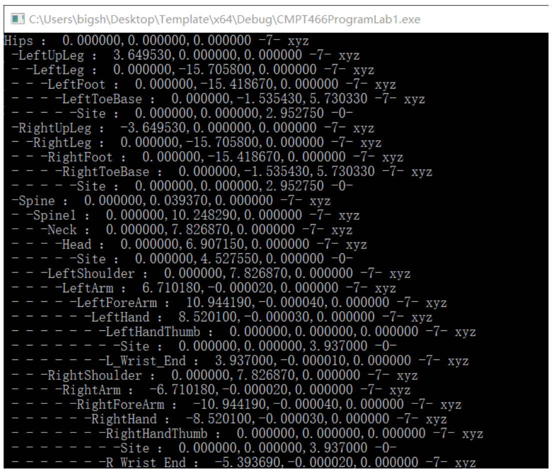
By default, the skeleton tree structure and motion data of the bvh file are loaded into an instance of the `ForwardKinematic` class (by `loadFile(const char* pfile)`), which contains a pointer (`Joint* root`) to the root joint of the skeleton tree, a pointer (`float** frameData`) to all frames' data, a pointer (`float* pFrame`) to one to-be-displayed frame data, etc. The overall workflow is as follows in the main function:



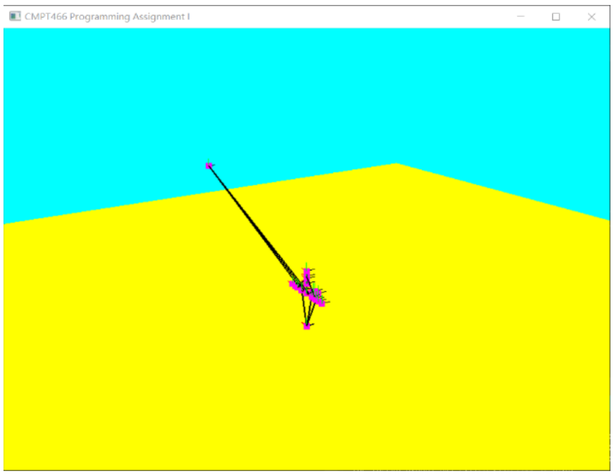
```
//create an instance of FK class
ForwardKinematic FK_Model;

//Main enntry
int main(int argc, char *argv[])
{
  //OpenGL initilize operations
  OpenGLinit(argc, argv);

  //Load bvh file and reconstruct the skeleton
  FK_Model.load_BVH("../data/running.bvh");

  //Print the loaded human skeleton
  FK_Model.print();

  //While loop
  while (true)
  {
    //response to the keyboard event: presing "w", load next frame's data to 'pFrame'
    glutMainLoopEvent();

    //recalculate the skeleton joints' positions with new frame's data
    FK_Model.calculateJointPos();

    glutPostRedisplay(); //re-display the new skeleton
  }

  return 0;
}
```

1) Initialize OpenGL: `OpenGLinit(argc, argv)`
2) Load BVH file, generate a skeleton tree with a pointer to the root (Joint* root) and, store all motion data to 'Float** frameData': `FK_model.load_BVH("../data/running.bvh")`
3) Print the skeleton tree: `FK_Model.print()`.
4) Loopinng: `While (true)`
   a. Response to Keyboard event: press 'w', load next frame's data (`glutMainLoopEvent()`: `pFrame=frameData[currentFrame+1]`);
   b. recalculate every joint's global position of the skeleton with 'root' and 'pFrame' (`FK_model.calculateJointPos()`: call `calculateJointPosRecursivelyWithQuaternion (root)`);
   c. Re-display the newly calculated skeleton: `glutPostRedisplay()`.

By default, the template display the animation like the followings (run the project in Visual Studio 2017 and 64-bit compiler): print Skeleton structure in output window (left) and display skeleton animation in GLUT window (right). Each time 'w' on the keyboard is pressed, one more frame is displayed.



```
C:\Users\bigsh\Desktop\Template\x64\Debug\CMPT466ProgramLab1.exe
Hips :  0.000000,0.000000,0.000000 -7- xyz
 -LeftUpLeg :  3.649530,0.000000,0.000000 -7- xyz
 - -LeftLeg :  0.000000,-15.705800,0.000000 -7- xyz
 - - -LeftFoot :  0.000000,-15.418670,0.000000 -7- xyz
 - - - -LeftToeBase :  0.000000,-1.535430,5.730330 -7- xyz
 - - - - -Site :  0.000000,0.000000,2.952750 -0-
 -RightUpLeg :  -3.649530,0.000000,0.000000 -7- xyz
 - -RightLeg :  0.000000,-15.705800,0.000000 -7- xyz
 - - -RightFoot :  0.000000,-15.418670,0.000000 -7- xyz
 - - - -RightToeBase :  0.000000,-1.535430,5.730330 -7- xyz
 - - - - -Site :  0.000000,0.000000,2.952750 -0-
 -Spine :  0.000000,0.039370,0.000000 -7- xyz
 - -Spine1 :  0.000000,10.248290,0.000000 -7- xyz
 - - -Neck :  0.000000,7.826870,0.000000 -7- xyz
 - - - -Head :  0.000000,6.907150,0.000000 -7- xyz
 - - - - -Site :  0.000000,4.527550,0.000000 -0-
 - - -LeftShoulder :  0.000000,7.826870,0.000000 -7- xyz
 - - - -LeftArm :  6.710180,-0.000020,0.000000 -7- xyz
 - - - - -LeftForeArm :  10.944190,-0.000040,0.000000 -7- xyz
 - - - - - -LeftHand :  8.520100,-0.000030,0.000000 -7- xyz
 - - - - - - -LeftHandThumb :  0.000000,0.000000,0.000000 -7- xyz
 - - - - - - - -Site :  0.000000,0.000000,3.937000 -0-
 - - - - - - -L_Wrist_End :  3.937000,-0.000010,0.000000 -7- xyz
 - - -RightShoulder :  0.000000,7.826870,0.000000 -7- xyz
 - - - -RightArm :  -6.710180,-0.000020,0.000000 -7- xyz
 - - - - -RightForeArm :  -10.944190,-0.000040,0.000000 -7- xyz
 - - - - - -RightHand :  -8.520100,-0.000030,0.000000 -7- xyz
 - - - - - - -RightHandThumb :  0.000000,0.000000,0.000000 -7- xyz
 - - - - - - - -Site :  0.000000,0.000000,3.937000 -0-
 - - - - - - -R Wrist End :  -5.393690,-0.000020,0.000000 -7- xyz
```



By keeping pressing ‘w’, you will see one purple square point (the root joint) moving around while the rest stayed the same. The purple square points indicate joints. Tiny red, green, blue lines indicate local coordinate axises of each joint (by default, all of them are set to the same as the world coordinate system).

Note:

If you meet with following error:

```
Error  MSB8036   The Windows SDK version 10.0.16299.0 was not found.
```

Go to **Project->Property** Open the window, change **Configuration_Properties->General->Windows SDK Version** to change the SDK version to your VS2017

## Coding Part

### Part 1) ComputeLocalQuaternion(Joint* joint) (5 /15 points)

```
/*Coding Part: 1) calculate local rotation in quaternion from euler angles for current node*/
Vector4 localQuat = computeLocalQuaternion(joint);
```

```
Input:
  Joint* joint
Output:
  Vector4 Joint’s local rotation in quaternion
```

**Every three adjacent elements within `pFrame` indicate three rotation angles along local x, y, z axis, respectively, with a specific rotation order in Euler angle (opposite to fixed-angle rotation order).**

```
Examples:
Joint(Hips):
    Rotation_order: joint->rotationOrder (e.g., XYZ=7).
    Rotation_angles:
    X: pFrame[0] rotation angle along X axis,
    Y: pFrame[1] rotation angle along Y axis,
    Z: pFrame[2] rotation angle along Z axis.
Joint(LeftUpLeg):
    Rotation_order: joint->rotationOrder (e.g., ZYX=1).
    Rotation_angles:
     Z: pFrame[3] rotation angle along Z axis,
     Y: pFrame[4] rotation angle along Y axis,
     X: pFrame[5] rotation angle along X axis …
```

1) You can get current joint’s rotation order by `joiont->rotationOrder`. The rotation order is stored in an `enum` structure:

   ```
   enum { NONE = 0, ZYX = 1, YZX = 2, ZXY = 3, XZY = 5, YXZ = 6, XYZ = 7 };
   ```

   Note that when rotation order is **`NONE`**, you shouldn’t move **`pFrame`** forward.
2) You can get current rotation angle from `float* pFrame[0, 1, 2]`, which is global variable within scope of **`ForwardKinematic`**. (Note: don’t forget to update **`pFrame`** after getting current rotation angle, or you will get root rotation angle for all joints)

3) You need to transform angle from degree to rad by multiply **`3.14f/180.f`**

4) `Vector4` class is provided to represent the quaternion with the first three elements representing x, y, z values and the last element representing w value: `Vector4 quaternion=Vector4(x,y,z,w).`
5) You need to calculate the quaternion from the Euler angles, using function `quaternionMultiplication(Vector4, Vector4)` .

If you complete part I correctly you would see **each joint’s local coordinate system is rotating** by pressing ‘w’.

## Part 2) `computeGlobalQuaternion(Joint* joint, Vector4 localQuat)` (5 /15 points)

```
Input:
    Joint* joint, Vector4 localQuat
Output:
    Vector4 joint's global rotation in quaternion
```

The local rotation for each joint is respect to its parent. So the global rotation for one joint is actually accumulated from the root.

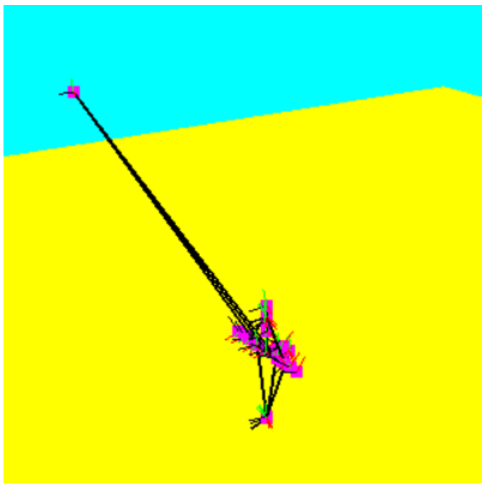
```
/*Coding Part: 2) calculate global rotation quaternion for child nodes*/
joint->Globalquat = computeGlobalQuaternion(joint, localQuat);
```

You need to accumulate all the quaternions along one specific path in the tree from root to current accessed joint (composite the quaternions!).

(Note: Remember you are coding in a recursive function, so you just need to deal with **current joint's local quaternion** and **parent joint's global quaternion**).

(Note: Be careful with root)

If part 2 is completed, you would see something similar after completing the Part1 (the local coordinate systems would be slightly different). But you know the rotation is accumulated!



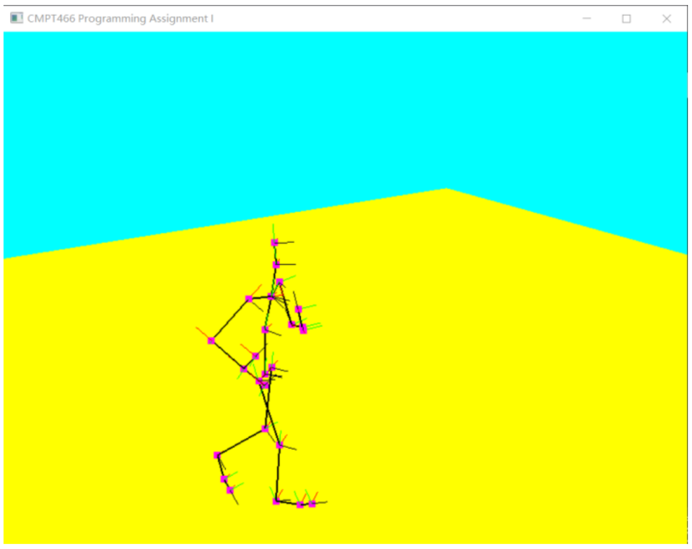
## Part 3) `computeGlobalPosition(BVHJoint* joint)` (5 /15 points)

```
/*Coding Part: 3) calculate current node's global position*/
Vector4 GlobalPosition = computeGlobalPosition(joint);
```

Based on part 1 and part 2, you now get the accumulated rotation from the root for each joint. The global position is calculated by parent's global position + rotated current joint's local position.

Again, it becomes easy as you are coding in a recursive function.

If you complete part 3, the result animation would be like a running skeleton by keeping pressing 'w' on keyboard.



(**Note: You may need to use global quaternion of its parent rather itself to compute relative positions**)

## Submittion

Please submit a zip file with name is your student number. The zip file will only contain your final modified ForwardKinematic.cpp file. Please make sure that the code is runnable in VS2017 community under Windows 10 environment before you submit it.

TA will test your code on the desktops in ASB9840 with VS2017.