PA1 – Programming Assessment

Due Date

- Assignment due on September 21, by Midnight
    - Following Morning
- Submit program to perforce in your student directory
    - Sub directory called: /PA1/…
    - Fill out your ***PA1 Submission Report.pdf***
        - Check it out and submit to same directory
        - There is nothing to Cut and Paste, leave it blank in the report
        - There are no tests to enter, leave it blank
        - Enter the final Changelist number of your PA1 submission

Goals

- Programming Assessment
    - Real world C++ exam that we give to interview candidates
    - Great practice

Assignments

- Write all programs in cross-platform C or C++.
    - ***Optimize for execution speed and robustness.***
- Create a programming file for each problem, for example
    - Student directory - for this assignment *only the CPP files submitted*
        - /PA1/problem1.cpp
        - /PA1/problem2.cpp
        - /PA1/problem3.cpp
        - /PA1/problem4.cpp
- Do all your work by yourself
    - Feel free to talk with others about setup, version control
    - ***DO NOT SHARE*** coding ideas on this assignment
    - Do not copy your friend’s code.
        - This is a competition!
- Check in the problems multiple times, at least 4 times for this PA (programming assignment)
    - Have reasonable check-in comments
- Make sure that your program compiles and runs
    - Warning level 4, some times that is not possible due to MS headers…
    - Your code should be squeaky clean.
    - There should be no warnings in the code that you did.
- Read the instructions carefully
    - Optimize all code for ***Speed*** and ***Robustness***
    - If you program doesn't compile, it's a zero.
    - If you submit a project make sure that minimum files are submitted
        - See perforce thread on how to submit minimum code
    - Most students do VERY poorly on this assignment
        - They are rushing and not thinking about this assignment correctly

- In the first class we go through all the optimizing ideas that you would need to successfully complete this assignment, please review
- C++ fundamentals sometimes trip up students
    - Think about the interfaces
        - Clean simple robust interfaces are needed
        - Think about the end user, how would they call the functions
    - Should you use pointers, references, or value?
    - What external functions are you calling?
        - Look up each function, make sure you understand what each function does
        - Think about how it's used from a performance point of view.
        - Can those routines give you errors?
- We are using Perforce
    - You should have received the document describing how to login.
        - Please look at the documentation on the class website
    - IP and Port of the server:
        - IP: 140.192.39.61
        - PORT: 1666
    - Submit program to perforce in your student directory
        - Sub directory called: /PA1/…
- There is a sample problem4.cpp in that directory that you can use as a reference to refactor. (saves typing)

Validation

Simple check list to make sure that everything is checked in correctly

- Did you do all 4 problems?
- Do they compile and run without any errors?
- Warning level 4 free?
- Submitted it into /PA1 directory?
- Can you delete you local drive, regrab the PA1 directory?
    - Is all the code there?

Hints

Most assignments will have hints in a section like this.

- Do many little check-ins
    - Iteration is easy and it helps.
    - Perforce is good at it.
- Think about performance, did you read the Code Complete chapters?
    - A lot of good ideas in there.

Write all programs in cross-platform C or C++.

- Optimize for execution speed and robustness.

1) ***(C/C++)*** Write a function to calculate this equation:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

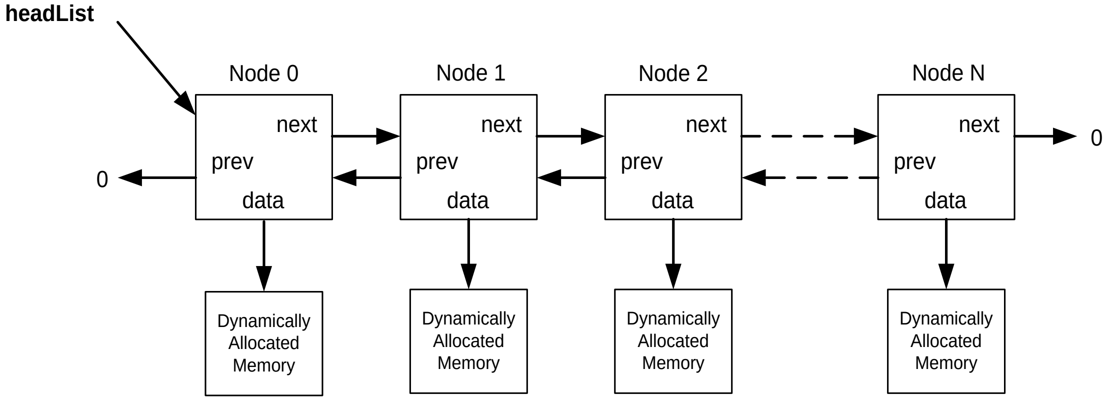
(Assume input: float x, output: float y.)

2) ***(C/C++)*** Write a function to normalize a 3 dimensional Cartesian vector.

3) ***(C/C++)*** Write a function that removes ***a given node*** in a linked list container containing nodes of type LinkList. Assume that each LinkList node and its data pointer were originally allocated dynamically by either malloc (C code) or new (C++ code).

```
typedef struct LinkList
{
      struct LinkList         *next;
      struct LinkList         *prev;
      struct ImportantData    *data;
} LinkList_t;

LinkList_t *headList;
```



4) ***(C/C++)*** The findMaxDistance() function (given below) calculates the maximum distance between any two players in the passed-in player array. Refactor it so that it ***also*** calculates the minimum distance between any two players. You may change the signature of the function and the contents of the function, but do not change the Vect_t structure.

```
typedef struct Vect  // Vector struct for positions
{
   float x;
   float y;
   float z;
} Vect_t;
```

```
/***********************************************************************
*
*  Function: findMaxDistance()
*
*  Input:
*          int       nPlayers - number of players
*          Vect_t   *playerArray - the array of players
*  Output:
*          float     maxDist - the maxDistance between any two players
*
***********************************************************************/

float findMaxDistance( int nPlayers, Vect_t *playerArray )
{
     int i,j;          // counter variables
     Vect_t tmpVect;   // temporary vector
     float tmpDist;    // temporary distance
     float maxDist;    // current max distance

     // initialize the distance to zero
     maxDist = 0.0f;

     for( i = 0; i < nPlayers; i++ )
        {
        for( j = 0; j < nPlayers; j++ )
           {
           // Find a vector between point i and j
           tmpVect.x = playerArray[i].x - playerArray [j].x;
           tmpVect.y = playerArray[i].y - playerArray [j].y;
           tmpVect.z = playerArray[i].z - playerArray [j].z;

           // Get its length
           tmpDist = (float)sqrt( tmpVect.x * tmpVect.x
                                  + tmpVect.y * tmpVect.y
                                  + tmpVect.z * tmpVect.z );

           // determine if it’s a new maximum length
           if( tmpDist > maxDist)
              {
              // yes so keep it
              maxDist = tmpDist;
              }

           } //for(j)
        } // for(i)

       return maxDist;

} // End of findMaxDistance()
```