

Jul 16, 02 18:16

README

Page 1/5

C++ Computational Steering Library:
=====

Download URL: <http://www.anirudh.net/phd/steering.tgz>
OR <http://bart.ihpca.psu.edu/phd/steering.tgz>

This distribution contains libraries for HP-UX (using aCC), SUN (using g++), Windows NT/2000 (using CYGWIN g++), and Linux (using g++).

There is a client/server demo app in the demo/ directory. Just run the appropriate Makefile using "make -f Makefile.PLATFORM" in each of the "client" and "server" directories to get the respective binaries.

-Anirudh Modi- <anirudh@anirudh.net>
10/12/2001-Fri

Registration functions:
=====

1. Only "global" data can be registered to be served by the server.
2. Include the header "dataserver.h" in the C++ code containing main().
3. All registration has to be specified within the function to be defined by the user:

```
REGISTER_DATA_BLOCK()
{
/* All REGISTER_WHATEVER commands come here */
}
```

4. Just invoke the Data Server by issuing an initialization to the DataServer class in the main() function.
e.g., DataServer Server; // Run on default port 4096

Optionally, the port number for the server can be specified.
e.g., DataServer Server(5000); // Run on port 5000

Another way to run the invoke the server is:

```
DataServer *Server = new DataServer;
if (Server->Start(5000) != CSL_SUCCESS)
{
/* Server did not start! Your Error Message Here */
}
```

This is recommended as you can get the status of the connection.

5. Here are the functions/macros provided to register various data types:
 - a. To register single variable (any type provided by the system, including int, short, long, float, double, char):

```
REGISTER_VARIABLE(key, "rw", variable);
```

e.g. REGISTER_VARIABLE("time", "ro", dt); // dt is defined as double

This function is independent of byte ordering (endian-ness) which may possibly be different from machine to machine (server/client).

The second parameter specifies the permission for the variable registered. "rw" means that the variable is in READ-WRITE mode and can be modified by the client. "ro" is READ-ONLY mode in which the client can only view the variable and not modify it. This is true for all the REGISTER macros.

- b. To register static arrays of any system type (int, double, ...), use:

```
REGISTER_1D_ARRAY(key, "rw", array1D);
REGISTER_2D_ARRAY(key, "rw", array2D);
REGISTER_3D_ARRAY(key, "rw", array3D);
REGISTER_4D_ARRAY(key, "rw", array4D);
```

NOTE: The arrays should all be static. i.e, "double mat[3][4]" can be registered, but not "double **mat" which may possibly be allocated the same size dynamically using malloc/new.

e.g. REGISTER_2D_ARRAY("matrixA", "rw", A); // A is defined as float A[2][9];

This function is independent of byte ordering (endian-ness) which may possibly be different from machine to machine (server/client).

- c. To register dynamic arrays of any system type (int, double, ...), use:

```
REGISTER_DYNAMIC_1D_ARRAY(key, "rw", array1D, dim1);
REGISTER_DYNAMIC_2D_ARRAY(key, "rw", array2D, dim1, dim2);
REGISTER_DYNAMIC_3D_ARRAY(key, "rw", array3D, dim1, dim2, dim3);
REGISTER_DYNAMIC_4D_ARRAY(key, "rw", array4D, dim1, dim2, dim3, dim4);
```

NOTE: The arrays should all be dynamic. i.e, "double **mat" can be registered, but not "double mat[3][5]" which may possibly be allocated the same size. dim1, dim2, ... denote the number of elements in each dimension. They may be variable or constant, but have to be defined as global variables. The following registration is illegal:

i.e, REGISTER_DYNAMIC_3D_ARRAY("dyn3D", "ro", dyn3D, n1, 7, n3); // Illegal

Also "double **mat; // ALLOC2D(&mat, n1, n2)" is a 2D dynamic array whereas "double (*mat)[5]; // ALLOC1D(&mat, n1)" is just a 1D dynamic array (as only one dimension is dynamic, the other is static).

```
REGISTER_DYNAMIC_1D_ARRAY("dyn1D", "rw", dyn2D, n1);
REGISTER_DYNAMIC_4D_ARRAY("dyn4D", "rw", dyn4D, n1, n2, n3, n4);
```

This function is independent of byte ordering (endian-ness) which may possibly be different from machine to machine (server/client).

- d. To register a user defined structure, use:

```
REGISTER_STRUCTURE(key, "ro", mystruct); // myStruct mystruct;
```

Also, along with this, two additional functions, packStruct/unpackStruct has to be defined by the user:

```
-----
int packStruct(myStruct *S, unsigned char **dataptr, int &totsize);
-----
```

packStruct allocates "totsize" bytes of data to the address pointed

Jul 16, 02 18:16

README

Page 3/5

to by "dataptr" and packs the individual members of the structure into this contiguous memory. It returns the location of the block where the last data was written to.

e.g.:

```
int packStruct(myStruct *S, unsigned char **dataptr, int &totsize)
{
    totsize = /*...(user specified size for the structure)...*/;
    unsigned char *data = new unsigned char[*totsize];
    *dataptr = data;
    int ptr = 0;

    /* pack data from S starting at &data[ptr] and update ptr accordingly */
    PACK_VARIABLE(S->member1, data, &ptr);
    PACK_2D_ARRAY(S->member2, data, &ptr);
    .
    .
    .

    return ptr;
}
```

NOTE: "ptr" should always be equal to "*totsize", else a runtime error would be generated. This just serves as a simple check to ensure that the user indeed packs in the amount of data he claims to pack when defining "*totsize".

```
-----
int unpackStruct(myStruct *S, unsigned char *data, int size);
-----
```

unpackStruct unpacks "size" bytes of data from contiguous memory block "data" into the structure S (essentially filling the structure). It returns the location of the block from where the data was last read from.

```
int unpackStruct(myStruct *S, unsigned char *data, int size)
{
    int ptr = 0;

    /* pack data from S starting at &data[ptr] and update ptr accordingly */
    UNPACK_VARIABLE(&(S->member1), data, &ptr);
    UNPACK_2D_ARRAY(S->member2, data, &ptr);
    .
    .
    .

    return ptr;
}
```

NOTE: "ptr" should always be equal to "size", else a runtime error would be generated. This just serves as a simple check to ensure that the user indeed unpacks in the amount of data he claims to pack in the function packStruct();

Packing/Unpacking functions:
=====

Jul 16, 02 18:16

README

Page 4/5

All packing/unpacking functions are independent of byte-ordering.
 All the routines are independent of variable type (int, double, ...).
 All routines have the same last 2 arguments of the form:

```
PACK_WHATEVER(variable or array, unsigned char *data, int *ptr);
```

which means that pack "variable/array" starting location &data[*ptr] and update *ptr accordingly by incrementing it with the size of the "variable/array".

Here are all the routines:

```
PACK_VARIABLE(var, data, &ptr);
PACK_1D_ARRAY(array1D, data, &ptr);
PACK_2D_ARRAY(array2D, data, &ptr);
PACK_3D_ARRAY(array3D, data, &ptr);
PACK_4D_ARRAY(array4D, data, &ptr);
PACK_DYNAMIC_1D_ARRAY(array1D, num_elems, data, &ptr);
PACK_DYNAMIC_2D_ARRAY(array2D, num_elems, data, &ptr);
PACK_DYNAMIC_3D_ARRAY(array3D, num_elems, data, &ptr);
PACK_DYNAMIC_4D_ARRAY(array4D, num_elems, data, &ptr);
```

```
UNPACK_VARIABLE(&var, data, &ptr);          // Note the presence of "&"
UNPACK_1D_ARRAY(array1D, data, &ptr);
UNPACK_2D_ARRAY(array2D, data, &ptr);
UNPACK_3D_ARRAY(array3D, data, &ptr);
UNPACK_4D_ARRAY(array4D, data, &ptr);
UNPACK_DYNAMIC_1D_ARRAY(array1D, num_elems, data, &ptr);
UNPACK_DYNAMIC_2D_ARRAY(array2D, num_elems, data, &ptr);
UNPACK_DYNAMIC_3D_ARRAY(array3D, num_elems, data, &ptr);
UNPACK_DYNAMIC_4D_ARRAY(array4D, num_elems, data, &ptr);
```

Client-Side calling routines:

```
=====
```

Client is invoked by invoking the Class DataClient.

```
e.g., DataClient Client(name); // Connect to server "name" on port 4096
```

Optionally the port on which the server is running can be specified.

```
e.g., DataClient Client(name, 5000); // Connect to server "name" on port 5000
```

Here are the client-side routines:

```
a. RecvVariable(key, &variable);
   OR variable = RecvVariable<type>(key);
```

This is used for receiving any type of variable registered on the server side using the keyword/handle "key".

```
e.g., double dt;
      client.RecvVariable("time", &dt);
      OR dt = client.RecvVariable<double>("time");
```

```
Corresponding Send: client.SendVariable("time", dt);
                   OR client.SendVariable<double>("time", 0.1);
```

```
b. RecvArray(key, array);
```

Jul 16, 02 18:16

README

Page 5/5

This is used for receiving any type of static array (1D/2D/3D/4D) registered on the server side using the keyword/handle "key".

```
e.g., float A[20][30];
      client.RecvArray("matrix", A);
```

Corresponding Send: client.SendArray("matrix", A);

```
c. RecvArray1D(key, arr1d);
   RecvArray2D(key, arr2d);
   RecvArray3D(key, arr3d);
   RecvArray4D(key, arr4d);
   dimN = getArrayDim(key, N);
```

These are used for receiving any 1D/2D/3D/4D dynamic array registered on the server side using the keyword/handle "key". getArrayDim() gets the values of the corresponding dimensions for the dynamic array. N = 1, 2, ..., number of dimensions.

```
e.g., double **A;
      int n1 = client.getArrayDim("matrix", 1);
      int n2 = client.getArrayDim("matrix", 2);
      ALLOC2D(&A, n1, n2);
      client.RecvArray2D("matrix", A);
```

Corresponding Send: client.SendArray2D("matrix", A);

Optionally, address of the array can also be passed if the programmer wants the library to automatically allocate the correct dimensions to the dynamic array.

```
e.g., double **A;
      int n1 = client.getArrayDim("matrix", 1);
      int n2 = client.getArrayDim("matrix", 2);
      client.RecvArray2D("matrix", &A);
```

Note, there is no ALLOC2D call needed for allocation in the above example, as RecvArray2D will automatically allocate the right amount of memory for A.

```
d. RecvStruct(key, &mystruct)
```

This is used for receiving any custom structure which has been registered on the server side using REGISTER_STRUCTURE command. packStruct()/unpackStruct() need to be defined for this structure.

Corresponding Send: client.SendStruct("matrix", &A);

```

//*****
#include <math.h>
#include <string.h>
#include "struct.h"
#include "alloc.h"
#include "dataserver.h"
//*****
// Define all data that has to be registered for being served as "global"
int run_flag = 0;
double mat[10][10];
char str[200];
char str2[200];
int N;
int i;
int mycount = 1;
int intarray[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
double dd[7];
double test_double = -4.32e+300;
float test_float = 1.234;
double *dyn;
int **dyn2D;
short ***dyn3D;
float ****dyn4D;
int n1, n2, n3, n4;
Trajectory T;
//*****
// Register all the variables/data here along with their handles/keywords
REGISTER_DATA_BLOCK()
{
REGISTER_VARIABLE("run_flag", "rw", run_flag);
REGISTER_VARIABLE("i", "rw", i);
REGISTER_VARIABLE("count", "rw", mycount);
REGISTER_VARIABLE("double", "ro", test_double);
REGISTER_VARIABLE("float", "ro", test_float);
REGISTER_1D_ARRAY("string", "rw", str);
REGISTER_1D_ARRAY("string2", "rw", str2);
REGISTER_1D_ARRAY("intarray", "ro", intarray);
REGISTER_1D_ARRAY("dd", "ro", dd);
REGISTER_2D_ARRAY("mat", "ro", mat);
REGISTER_DYNAMIC_1D_ARRAY("dyn", "rw", dyn, N);
REGISTER_DYNAMIC_2D_ARRAY("dyn2D", "rw", dyn2D, n1, n2);
REGISTER_DYNAMIC_3D_ARRAY("dyn3D", "rw", dyn3D, n1, n2, n3);
REGISTER_DYNAMIC_4D_ARRAY("dyn4D", "rw", dyn4D, n1, n2, n3, n4);
REGISTER_STRUCTURE("struct", "rw", T);
}
//*****
int main(int argc, char *argv[])
{
int port = 4096;
if (argc > 1)
port = atoi(argv[1]);

DataServer *Server = new DataServer;

while (Server->Start(port) != CSL_SUCCESS) { }

int j, k, l;

i = 0;
Server->Wait("dd");
for (j = 0; j < 7; j++)
dd[j] = i%1000+(j+1)/10.0;
Server->Post("dd");
}

```

```

T.i = 200;
T.d = 0.5;
T.a[23][79] = 2379.1;

N = 28;
n1 = 3;
n2 = 4;
n3 = 5;
n4 = 2;

ALLOC1D(&dyn, N);
ALLOC2D(&dyn2D, n1, n2);
ALLOC3D(&dyn3D, n1, n2, n3);
ALLOC4D(&dyn4D, n1, n2, n3, n4);

for (i = 0; i < n1; i++)
for (j = 0; j < n2; j++)
    dyn2D[i][j] = i+j;

for (i = 0; i < n1; i++)
for (j = 0; j < n2; j++)
for (k = 0; k < n3; k++)
    dyn3D[i][j][k] = i+j+k;

for (i = 0; i < n1; i++)
for (j = 0; j < n2; j++)
for (k = 0; k < n3; k++)
for (l = 0; l < n4; l++)
    dyn4D[i][j][k][l] = (i+1)*100+(j+1)*10+(k+1)*1+(l+1)/10.0;

i = 0;
str2[0] = '\0';
while (!run_flag) mySleep(100*1000);           // Sleep for 100 ms

cout << "Starting the computation..." << endl;

while (run_flag)
{
    static int first_time = 0;
    int prev_time, cur_time;
    if (first_time == 0) cur_time = myTimer();
    first_time = 1;

    ++i;
    if (i%100000 == 0)
    {
        prev_time = cur_time;
        cur_time = myTimer();
        cout << "i = " << i << " (" << cur_time-prev_time << " ms)" << endl << flush;
    }

    // This blocks the DataServer from serving data registered by
    // keyword "mat" when between the commands Wait() and Post().
    // Try commenting out the semaphore statements and running the
    // client....you will notice that value of "mat" differs!!
    Server->Wait("mat");
    for (j = 0; j < 10; j++)
    for (k = 0; k < 10; k++)
        mat[j][k] = (i%10)+j/10.0+k/100.0;
    Server->Post("mat");

    Server->Wait("dd");
    for (j = 0; j < 7; j++)

```

```

        dd[j] = i%1000+(j+1)/10.0;
Server->Post("dd");

Server->Wait("dyn");
for (j = 0; j < N; j++)
    dyn[j] = i%10+(j+1)/100.0;
Server->Post("dyn");

sprintf(str, "Test String! i = %d, mat[1][1] = %g", i, mat[1][1]);

if (str2[0] && mycount == 0)
{
    cout << "Received str2 = " << str2 << endl;
    ++mycount;
}

cout << "Stopping the computation." << endl;

delete Server;

FREE1D(&dyn, N);
FREE2D(&dyn2D, n1, n2);
FREE3D(&dyn3D, n1, n2, n3);
FREE4D(&dyn4D, n1, n2, n3, n4);
}
/*****/

```



```

//*****
#include <math.h>
#include <iomanip.h>
#include "struct.h"
#include "dataclient.h"
//*****
int main(int argc, char *argv[])
{
    char servername[100];
    int port = 4096;
    if (argc < 2)
    {
        cout << "Usage: client <server-name> [port-#]" << endl;
        exit(-1);
    }
    strcpy(servername, argv[1]);
    if (argc > 2)
        port = atoi(argv[2]);

    DataClient *client = new DataClient;

    if (client->Connect(servername, port) != POSSE_SUCCESS)
    {
        cout << "Connection to " << servername << ":" << port << " failed!" << endl;
        delete client;
        exit(-1);
    }

    client->SendVariable<int>("run_flag", 1);
    cout << "test_double = " << client->RecvVariable<double>("double") << endl << flush;
    cout << "test_float = " << client->RecvVariable<float>("float") << endl << flush;

    int t1 = myTimer();
    for (int ii = 0; ii < 5; ii++)
    {
        float test_float = client->RecvVariable<float>("float");
        cout << "Receiving test_float..ii = " << ii << endl << flush;
    }
    int t2 = myTimer();
    cout << "Total time consumed = " << t2-t1 << " ms." << endl << flush;

    char str[200];
    client->RecvArray("string", str);
    cout << "str: " << str << endl << flush;

    client->SendArray("string2", str);
    client->SendVariable("count", 0);

    int intarray[10];
    client->RecvArray("intarray", intarray);

    int i, j, k, l;
    for (i = 0; i < 10; i++)
        cout << "intarray[" << i << "] = " << intarray[i] << endl;

    double mat[10][10];
    client->RecvArray("mat", mat);
    for (i = 0; i < 10; i++)
    {
        for (j = 0; j < 10; j++)
            cout << " " << setprecision(4) << mat[i][j];
        cout << endl;
    }
}

```

```

    }

    int N = client->getArrayDim("dyn", 1);
    cout << endl << "N = " << N << endl;
    double *dyn;
    ALLOC1D(&dyn, N);
    client->RecvArray1D("dyn", dyn);

    for (i = 0; i < N; i++)
        cout << "dyn[" << i+1 << "] = " << dyn[i] << endl;
    cout << flush;
    FREE1D(&dyn, N);

    Trajectory a;
    client->RecvStruct("struct", &a);
    cout << "T.i = " << a.i << endl;
    cout << "T.d = " << a.d << endl;
    cout << "T.a[23][79] = " << a.a[23][79] << endl;
    ++a.i;
    client->SendStruct("struct", &a);

    int n1, n2, n3, n4;
    int **dyn2D;
    n1 = client->getArrayDim("dyn2D", 1);
    n2 = client->getArrayDim("dyn2D", 2);
    cout << "dyn2D = " << n1 << "x" << n2 << endl;
    client->RecvArray2D("dyn2D", &dyn2D);

    for (i = 0; i < n1; i++)
    {
        for (j = 0; j < n2; j++)
            cout << " " << dyn2D[i][j];
        cout << endl;
    }

    short ***dyn3D;
    n1 = client->getArrayDim("dyn3D", 1);
    n2 = client->getArrayDim("dyn3D", 2);
    n3 = client->getArrayDim("dyn3D", 3);
    client->RecvArray3D("dyn3D", &dyn3D);

    for (i = 0; i < n1; i++)
    for (j = 0; j < n2; j++)
    for (k = 0; k < n3; k++)
        cout << "dyn3D[" << i << "][" << j << "][" << k << "] = "
            << dyn3D[i][j][k] << endl;

    float ****dyn4D;
    n1 = client->getArrayDim("dyn4D", 1);
    n2 = client->getArrayDim("dyn4D", 2);
    n3 = client->getArrayDim("dyn4D", 3);
    n4 = client->getArrayDim("dyn4D", 4);
    ALLOC4D(&dyn4D, n1, n2, n3, n4);
    client->RecvArray4D("dyn4D", dyn4D);

    for (i = 0; i < n1; i++)
    for (j = 0; j < n2; j++)
    for (k = 0; k < n3; k++)
    for (l = 0; l < n4; l++)
    {
        cout << "dyn4D[" << i+1 << "][" << j+1 << "][" << k+1 << "]["
            << l+1 << "] = " << setprecision(4) << dyn4D[i][j][k][l] << endl;
    }

```

```

        ++dyn4D[i][j][k][l];
    }
    client->SendArray4D("dyn4D", dyn4D);

    FREE4D(&dyn4D, n1, n2, n3, n4);
    //client->SendVariable("run_flag", 0);
    delete client;
}
/*****/

```

```

/*****
#ifndef _MYSTRUCT_H
#define _MYSTRUCT_H
/*****
typedef struct
{
    int i;
    double d;
    double a[200][400];
} Trajectory;
/*****
int packStruct(Trajectory *T, unsigned char **dataptr, int &totsize);
int unpackStruct(Trajectory *T, unsigned char *data, int size);
/*****
#endif // _MYSTRUCT_H
/*****

```

```

/*****
#include <assert.h>
#include <iostream.h>
#include "struct.h"
#include "myendian.h"
*****/
int packStruct(Trajectory *T, unsigned char **dataptr, int &totsize)
{
    totsize = sizeof(T->i) + sizeof(T->d) + sizeof(T->a);
    unsigned char *data = new unsigned char[totsize];
    *dataptr = data;
    int ptr = 0;

    PACK_VARIABLE(T->i, data, &ptr);
    PACK_VARIABLE(T->d, data, &ptr);
    PACK_2D_ARRAY(T->a, data, &ptr);

    return ptr;
}
/*****
int unpackStruct(Trajectory *T, unsigned char *data, int size)
{
    int ptr = 0;

    UNPACK_VARIABLE(&(T->i), data, &ptr);
    UNPACK_VARIABLE(&(T->d), data, &ptr);
    UNPACK_2D_ARRAY(T->a, data, &ptr);

    return ptr;
}
*****/

```

```

//*****
#ifndef WAKEVORTEX_H
#define WAKEVORTEX_H
//*****
#include <vector>
#include <stdio.h>
#ifdef HPUX
#include <iostream.h>
#else
#include <iostream>
#endif
#include "alloc.h"
#include "wing.h"
#include "airplane.h"
#ifdef _WIN32
using namespace std;
#endif
//*****
enum
{
    VWING_STOP_TRACKING = 0,
    VWING_DELETE
};
//*****
class Vortex
{
private:
    friend ostream &operator << (ostream &Out, Vortex &V);
    void movewings(int wingID);
    void shiftwake(int wingID);
    void movewakes(int wingID);
    void wakevel(int wingID);
    void agewake(int wingID);
    void findUVW(double u0, double v0, double w0, double delta, double upmag, double x, double y, double z, double *uu, double *vv, double *ww);
    inline void inducedvel(Wing *wing, int i, int j, double *vx, double *vy, double *vz, int sign);
    inline void inducedvel(Wing *wing, int i1, int j1, int i2, int j2, double *vx, double *vy, double *vz, int sign, int sign2);
    void inducedvel(double x1, double y1, double z1, double x2, double y2, double z2, double x, double y, double z, double *velx, double *vely, double *vz);
    void Initialize();

public:
    int nwings;
    int ntotal;
    int nToBeDeleted;
    int nOutOfRange;
    int nwakemax;
    double temperature; // in Celcius
    double dt;
    double u0;
    double v0;
    double w0;
    double delta;
    double upmag;
    double gtime;
    double glimit;
    double max_tracking_distance; // in Kilometers
    int inducedvelocity_flag;
    int iter;
    int kn;
    int nvort;
    double curtime;
    vector<Wing> wing;

    Vortex(int n = 0);
    ~Vortex();
    void MarkAllDeleted();
    void Cleanup();

```

```

inline int NextWing(int i);
inline int NextInRange(int i);
inline int Begin(void);
inline int NextToBeDeleted();

Vortex &operator=(const Vortex &);

int AddWing(int allocate_flag = 1);
int AddWingWithoutAllocation();
void ReAlloc(int n);
void Allocate(int i);
void Compress();
void deleteWing(int i);
void undeleteWing(int i);
void stopTrackingWing(int i);
int CheckForOutOfRangeAircraft(double airport_center[3], int option);
void ReadInputFile(char *inpfile, int nwake_max);
double ComputeWake();
int ComputeWake(int wingID);
void WriteTecplotFile(int iter);
int AddNewWing(char *trajfile, char *name, char *type, double x0, double y0, double z0, double span, double area, double weight);
int AddRandomWing(char *trajfile, double x0, double y0, double z0);
int Copy(Vortex *dest);

};

// *****
inline int Vortex::NextWing(int i)
{
    int j = i+1;
    while ((j < nwings) && wing[j].ToBeDeleted) ++j;

    return j;
}

// *****
inline int Vortex::Begin(void)
{
    int j = 0;
    while ((j < nwings) && wing[j].ToBeDeleted) ++j;

    return j;
}

// *****
inline int Vortex::NextInRange(int i)
{
    int j = (i+1)%nwings;
    while ((j < nwings) && (wing[j].ToBeDeleted || wing[j].OutOfRange)) ++j;

    return (j%nwings);
}

// *****
inline int Vortex::NextToBeDeleted()
{
    int j = 0;
    while ((j < nwings) && (wing[j].ToBeDeleted == 0)) ++j;

    return j;
}

// *****
#include "packvortex.h"
// *****
#endif // WAKEVORTEX_H
// *****

```

```

/*****/
#include <assert.h>
#include <iostream.h>
#include "wakevortex.h"
#include "myendian.h"
/*****/
static int sizeofdouble = sizeof(double);
static int sizeofint = sizeof(int);
/*****/
int packStruct(Vortex *V, unsigned char **dataptr, int &totsize)
{
    int i;
    static Wing tempWing;
    static int sizeofwing = sizeof(tempWing.gwing) + sizeof(tempWing.ewing)
        + sizeof(tempWing.xwing_i) + sizeof(tempWing.DistFromAirport)
        + sizeof(tempWing.xwing) + sizeof(tempWing.vwing)
        + sizeof(tempWing.vwing_i) + sizeof(tempWing.trajectoryname)
        + sizeof(tempWing.span) + sizeof(tempWing.ID)
        + sizeof(tempWing.area) + sizeof(tempWing.weight)
        + sizeof(tempWing.OutOfRange) + sizeof(tempWing.nvort)
        + sizeof(tempWing.ToBeDeleted) + sizeof(tempWing.iter)
        + sizeof(tempWing.Name) + sizeof(tempWing.AircraftType);

    totsize = 6*sizeofint + 11*sizeofdouble;           // Size of vortex variables

    assert(V->nwings - V->nToBeDeleted - V->ntotal == 0);

    for (i = 0; i < V->nwings; i++)
    {
        Wing *W = &V->wing[i];
        if (W->ToBeDeleted)
        {
            totsize += sizeofint;
        }
        else
        {
            totsize += sizeofwing;
            totsize += sizeofdouble*(W->nvort*2*3);
            totsize += sizeofdouble*W->nvort;
        }
    }

    unsigned char *data = NULL;
    ALLOC1D(&data, totsize);
    *dataptr = data;
    int ptr = 0;

    PACK_VARIABLE(V->nwings, data, &ptr);
    PACK_VARIABLE(V->inducedvelocity_flag, data, &ptr);
    PACK_VARIABLE(V->nOutOfRange, data, &ptr);
    PACK_VARIABLE(V->nwakemax, data, &ptr);
    PACK_VARIABLE(V->iter, data, &ptr);
    PACK_VARIABLE(V->kn, data, &ptr);
    PACK_VARIABLE(V->dt, data, &ptr);
    PACK_VARIABLE(V->u0, data, &ptr);
    PACK_VARIABLE(V->v0, data, &ptr);
    PACK_VARIABLE(V->w0, data, &ptr);
    PACK_VARIABLE(V->delta, data, &ptr);
    PACK_VARIABLE(V->upmag, data, &ptr);
    PACK_VARIABLE(V->gtime, data, &ptr);
    PACK_VARIABLE(V->glimit, data, &ptr);
    PACK_VARIABLE(V->curtime, data, &ptr);

```



```

PACK_VARIABLE(V->max_tracking_distance, data, &ptr);
PACK_VARIABLE(V->temperature, data, &ptr);

for (i = 0; i < V->nwings; i++)
{
    Wing *W = &V->wing[i];
    PACK_VARIABLE(W->ToBeDeleted, data, &ptr);

    if (W->ToBeDeleted)
        continue;

    PACK_VARIABLE(W->ID, data, &ptr);
    PACK_VARIABLE(W->nvort, data, &ptr);
    PACK_VARIABLE(W->iter, data, &ptr);
    PACK_VARIABLE(W->OutOfRange, data, &ptr);
    PACK_VARIABLE(W->DistFromAirport, data, &ptr);
    PACK_VARIABLE(W->gwing, data, &ptr);
    PACK_VARIABLE(W->ewing, data, &ptr);
    PACK_VARIABLE(W->span, data, &ptr);
    PACK_VARIABLE(W->area, data, &ptr);
    PACK_VARIABLE(W->weight, data, &ptr);
    PACK_2D_ARRAY(W->xwing_i, data, &ptr);
    PACK_2D_ARRAY(W->vwing_i, data, &ptr);
    PACK_2D_ARRAY(W->xwing, data, &ptr);
    PACK_2D_ARRAY(W->vwing, data, &ptr);
    PACK_1D_ARRAY(W->trajectoryname, data, &ptr);
    PACK_1D_ARRAY(W->Name, data, &ptr);
    PACK_1D_ARRAY(W->AircraftType, data, &ptr);
    PACK_DYNAMIC_3D_ARRAY(W->xwake, W->nvort*2*3, data, &ptr);
    PACK_DYNAMIC_1D_ARRAY(W->gwake, W->nvort, data, &ptr);
}

return ptr;
}
/*****
int unpackStruct(Vortex *V, unsigned char *data, int size)
{
    assert(V != NULL);
    V->CleanUp();

    int ptr = 0;

    int nwings;
    UNPACK_VARIABLE(&nwings, data, &ptr);
    UNPACK_VARIABLE(&(V->inducedvelocity_flag), data, &ptr);
    UNPACK_VARIABLE(&(V->nOutOfRange), data, &ptr);
    UNPACK_VARIABLE(&(V->nwakemax), data, &ptr);
    UNPACK_VARIABLE(&(V->iter), data, &ptr);
    UNPACK_VARIABLE(&(V->kn), data, &ptr);
    UNPACK_VARIABLE(&(V->dt), data, &ptr);
    UNPACK_VARIABLE(&(V->u0), data, &ptr);
    UNPACK_VARIABLE(&(V->v0), data, &ptr);
    UNPACK_VARIABLE(&(V->w0), data, &ptr);
    UNPACK_VARIABLE(&(V->delta), data, &ptr);
    UNPACK_VARIABLE(&(V->upmag), data, &ptr);
    UNPACK_VARIABLE(&(V->gtime), data, &ptr);
    UNPACK_VARIABLE(&(V->glimit), data, &ptr);
    UNPACK_VARIABLE(&(V->curtime), data, &ptr);
    UNPACK_VARIABLE(&(V->max_tracking_distance), data, &ptr);
    UNPACK_VARIABLE(&(V->temperature), data, &ptr);

    int numdeleted = 0;

```

```

for (int i = 0; i < nwings; i++)
{
    V->AddWing();
    Wing *W = &V->wing[i];
    UNPACK_VARIABLE(&(W->ToBeDeleted), data, &ptr);

    if (W->ToBeDeleted)
    {
        ++numdeleted;
        W->DeAllocate(V->nwakemax);
        continue;
    }

    UNPACK_VARIABLE(&(W->ID), data, &ptr);
    UNPACK_VARIABLE(&(W->nvort), data, &ptr);
    UNPACK_VARIABLE(&(W->iter), data, &ptr);
    UNPACK_VARIABLE(&(W->OutOfRange), data, &ptr);
    UNPACK_VARIABLE(&(W->DistFromAirport), data, &ptr);
    UNPACK_VARIABLE(&(W->gwing), data, &ptr);
    UNPACK_VARIABLE(&(W->ewing), data, &ptr);
    UNPACK_VARIABLE(&(W->span), data, &ptr);
    UNPACK_VARIABLE(&(W->area), data, &ptr);
    UNPACK_VARIABLE(&(W->weight), data, &ptr);
    UNPACK_2D_ARRAY(W->xwing_i, data, &ptr);
    UNPACK_2D_ARRAY(W->vwing_i, data, &ptr);
    UNPACK_2D_ARRAY(W->xwing, data, &ptr);
    UNPACK_2D_ARRAY(W->vwing, data, &ptr);
    UNPACK_1D_ARRAY(W->trajectoryname, data, &ptr);
    UNPACK_1D_ARRAY(W->Name, data, &ptr);
    UNPACK_1D_ARRAY(W->AircraftType, data, &ptr);
    UNPACK_DYNAMIC_3D_ARRAY(W->xwake, W->nvort*2*3, data, &ptr);
    UNPACK_DYNAMIC_1D_ARRAY(W->gwake, W->nvort, data, &ptr);
}

V->ntotal -= numdeleted;
V->nToBeDeleted = numdeleted;

return ptr;
}
//*****

```