Project 1, due on 02/20.

Problem 1. Use the Demo_Pkg code to implement the add-first algorithm, which is to add a new node as the first node of the list. See Figure 1. After insertion, the new node is the "first" node in the list.

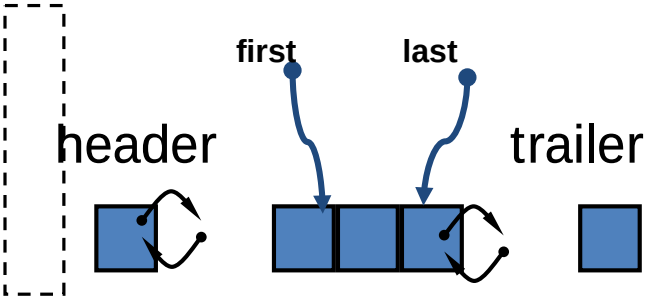


Figure 1.

The current code implements the add last algorithm in function insert_tri_at_tail_of_list(), which is to add a new node as the last node of the list.

Instructions:

1.1. Use the following declaration of the function to implement the add first algorithm:

LOCAL void insert_tri_at_head_of_list(TRI *tri, SURFACE *s);

Variable "tri" is the node to be inserted into the list maintained in "s".

1.2. Replace the insert_tri_at_tail_of_list() function by your insert_tri_at_head_of_list() function. In the main() function, use the following loop statement to print out ID of each of triangles in the list.

```
for (temp_tri = first_tri(outs); !at_end_of_tri_list(temp_tri,outs);
            temp_tri = temp_tri->next)
{
   printf("temp_tri ID = %d\n", temp_tri->id);
}
```

1.3 (Optional). Construct the adjacency of triangles by installing to Tri_on_side() of each of the triangles, the edge adjacent triangles. Compare your results with the adjacency information in newBurgers.e file.

Hint: this adjacency information can be retrieved from "neigh" argument of the read_input_easy_mesh().

Problem 2. Use the Demo_Pkg code to implement the add-first algorithm using the C++ STL List template.

2.1. In util/cdecs.h

```
#  include <list> /* For using STL List*/
```

2.2. In funcs_related_to_Proj1(), declare

```
list<TRI*> tri_list;
```

2.3. declare the following function in mesh_app/mprotos.h

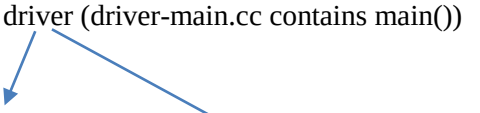
read_input_easy_mesh_using_STL_list("newBurgers", std::list<TRI*>&);

2.4. define read_input_easy_mesh_using_STL_list() in mesh_app/mesh.cc

Hand-In. Make a tar ball of all of your source code. Turn in the electronic and hardcopy of your report which contains results and pseudo-code description of your implementations. Email the tar ball of source code. Use the following title for your email: acms40212S15-Proj1-your-ND-ID

Code Structure of Demo_pkg:

driver (driver-main.cc contains main())



mesh_app(mesh.cc contains read_input_easy_mesh())

util (memory allocation, mpi utilities, IO)

```
/* Point in the Plane: */
struct _POINT{};

#define Coords(p)       ((p)->_coords)
#define COORDS(P)       ((P)._coords)

/* Triangle */
struct _TRI
{
       POINT *__pts[3];
};
#define Point_of_tri(_tri_)     (_tri_)->__pts
#define Tri_neighbor(_tri_)         (_tri_)->neighbor
#define Tri_on_side01(_tri_)        (Tri_neighbor(_tri_)[0].tri)
#define Tri_on_side12(_tri_)        (Tri_neighbor(_tri_)[1].tri)
#define Tri_on_side20(_tri_)        (Tri_neighbor(_tri_)[2].tri)
#define Tri_on_side(_tri_,side)      ((Tri_neighbor(_tri_) + (side))->tri)
```

scalar(a,b): allocate memory of size “b” byte. “a” holds the address of the allocated memory.

vector(a,b,c): allocate memory for a contiguous array of “b” elements. The size of an element is “c” byte. “a” holds the starting address of the allocated memory.

matrix(a,b,c,d): allocate memory for a 2D array of “b×c” elements. The size of an element is “d” byte. “a” holds the starting address of the allocated memory.

readline(): read a line of string of characters from input file

findfield(): return 0-9 character field in a string.

i_make_tri(POINT*p0,POINT*p1,POINT*p2,POINTER,POINTER,POINTER,int): create a TRI with 3 vertices given by p0, p1, p2.

Program flow:

```
read_input_easy_mesh()
{
      SURFACE    *s; // the surface pointer which holds the linked list
      TRI        **tri, *tmptri; // **tri is initialized as an array of TRI* pointers.
                          // Each entry of "tri" points to a TRI element
      POINT      **p, *pt[4], *tmpp[3]; //**p is initalized as an array of POINT* pointers.
                               // Each entry of "p" points to a POINT.

      /* init. surface and setup  the dummy nodes of the doubly linked list*/
      scalar(&s, sizeof(SURFACE));
      first_tri(s) = tail_of_tri_list(s);
      last_tri(s) = head_of_tri_list(s);
      s->num_tri = 0;

       /* Create points of the mesh */
        for(i = 0; i < n_node; i++)
      {
         coords[0] = crds_node[i][0];
         coords[1] = crds_node[i][1];
         coords[2] = 0.0;
         p[i] = Point(coords,3);
      }

       /* create triangle and add it to the linked list */
       for(i = 0; i < n_tri; i++)
      {
         pt[0] = p[tri_node[i][0]];
         pt[1] = p[tri_node[i][1]];
         pt[2] = p[tri_node[i][2]];

         tri[i] = i_make_tri(pt[0],pt[1],pt[2],
                          NULL,NULL,NULL,YES);
         tri[i]->id = i;
         insert_tri_at_tail_of_list(tri[i],s);
      }

       /*Pass the mesh information back to caller of this function */
       *out_neigh = neigh;
      *outs = s;
      *tri_num = n_tri;

       return tri;
}
```

util/

cdecs.h: all standard C, C++ headers are included here. All machine dependent functions are declared here.

vmalloc.h: Definitions for Storage Allocators.

uprotos.h: Function prototypes for util library.

vectormalloc.c: Definition of storage functions.

ppsub.c: wrappers of MPI parallel communication functions.

mesh_app/

int.h: POINT, TRI, SURFACE are defined here.

comput_1d.h: Compute_1d class is defined here.

mprotos.h: Function prototype declaration for mesh_app library.

driver/

driver-main.cc: Function main() is here.

Add new files in mesh_app/

In mesh_app/makefile, add the name of the source and header files.

Instruction for installing and compiling the code is in the README.txt file.

README.txt is under ~zxu2/Public/Demo_pkg.