12.010 Computational Methods of Scientific Programming

Lecture 16: C/C++

Summary

- Programming concepts and syntax in C/C++
  - Some simple examples
  - Look at example numerical codes in C
    - Polyarea
  - Delve into use of
    - Pointers
    - Structures

C syntax in action

- Simple program exercise that shows functions, loops, pointers and copies etc…
  - Remember

```
[1]: y=2
     x=2+y
     x-y

[1]: 2

[2]: y=1.e23
     x=2+y
     x-y

[2]: 0.0
```

How does this look in C

```
#include <stdio.h>
int main(int argc, char *argv){
        double x,y;
        y=2.;
        x=y+2.;
        printf("%f\n",x-y);
}
```

```
cnh@mit.edu@ip-172-30-1-201:~$ gcc lec015_xplusy.c
cnh@mit.edu@ip-172-30-1-201:~$ ./a.out
2.000000
cnh@mit.edu@ip-172-30-1-201:~$
```

Include standard lib header file (for I/O)

Executable statements in a function `int main(…){…}`

Variables all need to be typed explicitly.

Statements have to end with `;`

Call a standard lib func to do I/O

Compile and the execute in terminal.

C syntax in action

- Lets create a function “below_epsilon” that returns true is a number is too small to change one when subtracted. Historically this is known as “machine epsilon”.

```c
#include <stdio.h>
int below_epsilon( double );
int main(int argc, char *argv){
        double x = 100.;
        if ( below_epsilon(x) ) {
         printf("1-x=1, for x =%f\n",x);
        } else {
         printf("1-x=1-x, for x =%f\n",x);
        }
}
```

Lec17_below_epsilon_main.c

```c
int below_epsilon(double x ){
        int is_below_eps=0;   // 0 i
                              // non
        double a, b;
        a = 1.;
        b = 1. - x;
        is_below_eps = (a == b);
        return is_below_eps;
}
```

Lec17_below_epsilon_func.c

- when we split in two files, need a “function prototype” in main, to give function and args types.

C syntax in action

- Are we sure 0 is false? We can write a program to test this.

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int a = 0;
    if ( a ) {
      printf("%d is true\n",a);
    } else {
      printf("%d is false\n",a);
    }
}
```

gcc Lec17_is_0_false_arg.c
./a.out to see

C syntax in action

- Are we sure 0 is false? We can loop to check a few values.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
   for (int a=-50;a<50;a=a+10) {
     if ( a ) {
       printf("%d is true\n",a);
     } else {
       printf("%d is false\n",a);
     }
   }
}
```

```
% gcc Lec17_is_0_false_loop.c
% ./a.out
-50 is true
-40 is true
-30 is true
-20 is true
-10 is true
0 is false
10 is true
20 is true
30 is true
40 is true
```

for (initial;test; increment) { body… }
if (condition) { expression.. } else { expression.. }

C syntax in action

- Are we sure 0 is false? We can check values passed as arguments

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
   if ( argc != 2 ) {
           printf("Error\n");
           exit(-1);
   }
   int a;
   int nmatch;
   nmatch = sscanf(argv[1],"%d",&a);
   if ( nmatch != 1 ) {
           printf("Error\n");
           exit(-1);
   }
   if ( a ) {
     printf("%d is true\n",a);
   } else {
     printf("%d is false\n",a);
   }
}
```

```
TAHMac[1548] cc  Lec17_is_0_false_arg.c
TAHMac[1549] a.out 300
300 is true
TAHMac[1550] a.out 0
0 is false
```

C syntax in action

- Try

Lec17_xplusy.c - Modify to demonstrate “eps”

Lec17_is_0_false.c - Modify to take an argument

Lec17_below_epsilon_main.c +
Lec17_below_epsilon_func.c

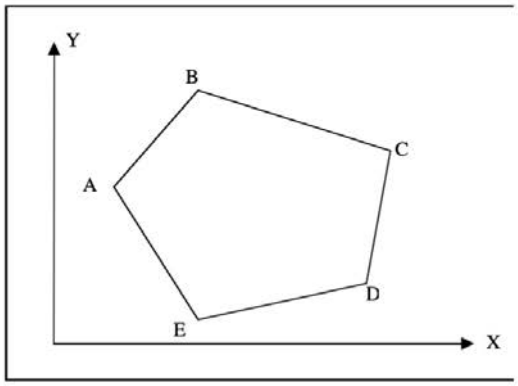
- Modify to take an argument and finds its eps with a loop.

C polyarea

Remember – polyarea.ipynb, Lec 04

How does this look in C.

- **Problem:** Find the area of an arbitrarily shaped plane figure.
- Figure defined by X,Y coordinates of vertices



C polyarea

Remember – polyarea.ipynb, Lec 04

How does this look in C.

~3 core functions, readnodes(), form_vectors(), triarea().
In C there is not built-in way to plot.

Lets start with high level

```
# Read in the polygon coordinates
print('\n POLYGON AREA calculation
numnode, nodes_xy = readnodes('pol
# (Substract one on output becuase
# same so that the line closes)
print('Polygon has',numnode-1,'nod
# Form the vectors from the first
trivec = form_vectors(nodes_xy)
# Compute the area of the polygon
Area = triarea(trivec)
# Print out the results
print('AREA of the polygon is ',Ar
# Plot the polygon
plotpoly(nodes_xy)
```

C polyarea

```
# Read in the polygon coordinates
print('\n POLYGON AREA calculation
numnode, nodes_xy = readnodes('pol
# (Substract one on output becuase
# same so that the line closes)
print('Polygon has',numnode-1,'nod
# Form the vectors from the first
trivec = form_vectors(nodes_xy)
# Compute the area of the polygon
Area = triarea(trivec)
# Print out the results
print('AREA of the polygon is ',Ar
# Plot the polygon
plotpoly(nodes_xy)
```

A basic poly_area main program in C

"main"

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int NMAX=1000;

/* Function prototypes */
int    read_nodes( double [2][NMAX], int );
void   get_tri(int, double [2][2], double [2][NMAX], int);
double calc_da(double [2][2]);
void   check_dir(int, int *, double);

int main(int argc, char *argv[]){
  /* Declare main variables
  double nodes_xy[2][NMAX];
  int    n_nodes;
  double tri_vec[2][2];
  double area, darea;
  int    sign_da;

  /* Start message */
  printf("=== Computing polygon area ==\n");

  /* Get node list */
  n_nodes=read_nodes(nodes_xy, NMAX);

  /* Check we have enough nodes */
  if ( n_nodes < 3 ) {
    printf("ERROR: Not enough nodes entered\n");
    exit(-1);
  }

  for (int i=2;i<n_nodes;i=i+1) {
    /* Form a triangle */
    get_tri(i,tri_vec,nodes_xy,n_nodes);
    /* Calculate triangle area increment */
    darea=calc_da(tri_vec);
    /* See if we are moving in consistent direction */
    check_dir(i,&sign_da,darea);
    /* Accumulate area */
    area=area+darea;
  }

  printf("Area = %lf\n",fabs(area));
}
```

Headers for "printf", "scanf", "exit", "fabs".

Global, static max size, later we will use pointer.

Types for our internal functions.

main function – everything starts here.

declare types of main variables

internal function to read

internal function to form triangles

internal triangle area function

internal direction check

sum area

print total area

C polyarea vs Python

```python
def readnodes( file='stdin' ):
    '''Function to read the X,Y coordinates of nodes of the
    points in the triangle.  The circuit around the nodes
    should be in clockwise direction.
    Input "file" is name of file.  If not passed, keyboard
    is used
    Useage:
    numnode, nodes_xy = readnodes('stdin'/file)

    '''

    # First see if file or std in to be read
    if file == 'stdin':
        inpstr = input('Coordinate pairs of nodes ')
        nodelist = list(map(float,inpstr.split(' ')))
        # Now make numpy array and reshape
        numlist = len(nodelist)
        if numlist != (numlist//2)*2:
            print('Even number of nodes nodes needed\n',numlist,'given')
            return 0, 0

        numnode = int(numlist//2)
        nodes_xy = np.array(nodelist).reshape(numnode,2)
    else:
        try:
            nodes_xy = np.genfromtxt(file,delimiter=',')
            numnode = np.shape(nodes_xy)[0]
        except:
            print('Exception reading',file)
            return 0, 0
    # Replicate the last element in array to close the polygon
    if not np.all(nodes_xy[numnode-1,:] == nodes_xy[0,:]):
        nodes_xy = np.append(nodes_xy,nodes_xy[0])
        # The matrix reverts to an array when this is done, so
        # a reshape is needed
        numnode += 1
        nodes_xy = np.array(nodes_xy).reshape(numnode,2)

    # Returns number of nodes and np array

    return numnode, nodes_xy
```

```c
int read_nodes(double n_xy[2][NMAX], int nmax){

    double X,Y,nread;
    int n_nodes=0;
    printf("Enter space-separated pairs of coordinates of the

    while (  ( nread=scanf("%lf %lf\n",&X,&Y) ) != -1 ) {
      if ( nread != 2 ) {
              printf("Invalid line format entered.\n");
              exit(-1);
      } else {
        n_xy[0][n_nodes]=X;
        n_xy[1][n_nodes]=Y;
        n_nodes=n_nodes+1;
      }
      if ( n_nodes == nmax ) {
              printf("Too many nodes.\n");
              exit(-1);
      }
    }
    printf("Polygon has %d nodes.\n",n_nodes);
    return n_nodes;
}
```

Pass in array to fill, return number of nodes read, error checking but no error recovery.

while loop testing for “end of file” (-1). if () {} -

I/O

C polyarea vs Python

```python
def form_vectors(nodes_xy):
    '''Function form the two vectors that make up the triangle
    from the node coordinates.
    Usage:
    trivec = form_vector(nodes_xy)
    where nodes_xy is numpy rows*2 array
    trivec is numpy array
    '''
    return nodes_xy-nodes_xy[0]
```

Python form all triangles in np array

```c
void get_tri(int i, double t_vec[2][2], double n_xy[2][NMAX], int n_n){
        t_vec[0][0]=n_xy[0][i-1]-n_xy[0][0];  /* Poly start to point 1 */
        t_vec[1][0]=n_xy[1][i-1]-n_xy[1][0];
        t_vec[0][1]=n_xy[0][i  ]-n_xy[0][0];  /* Poly start to point 2 */
        t_vec[1][1]=n_xy[1][i  ]-n_xy[1][0];
}
```

C loop over triangles, form one at a time for node "i".

Type declarations needed for every variable (NMAX is global).

Result is passed out through array (array is "pointer" to memory).

triangle (wrt to first point)

C polyarea vs Python

Python cross product and convex check

```python
def triarea(xy,scale=1.0):
    '''Compute the area enclosed by summing the
    each triangle that make up the polygon
    Useaage
    area = triarea(nodes_xy,scale)
    '''
    # Form the cross product Z-component and div
    # cross product a x b = |a||b| sin(theta) in
    # to the plane of vectors a and b.  theta is
    # vectors.
    # Note: Sign will depend on if we rotate clo
    # anti-clockwise between vector.  A change i
    # a change from convex to concave.
    # Numpy has method to form cross product
    n = np.shape(xy)[0]-2  # Take vectors in pai

    area = 0
    for k in range(1,n):
        darea = (xy[k,0]*xy[k+1,1]-xy[k+1,0]*xy[
        if k == 1:
            signa = np.sign(darea)
        else:
            if signa != np.sign(darea):
                print('Concave at node ',k)

        area += darea*scale**2

    return area
```

```c
double calc_da(double t_vec[2][2]){
        double da=0.;
        da = ( t_vec[0][0]*t_vec[1][1] -
               t_vec[1][0]*t_vec[0][1] )*0.5;
        return da;
}
```

cross-product area, $(a_x b_y - a_y b_x)*0.5$

```c
void check_dir(int inum,int *s, double da){
      if ( inum == 2 ) {
        if ( da < 0. ) {
         *s = -1;
        }
        else {
         *s = +1;
        }
      }
      else
      {
        if ( *s*da < 0. ) {
         printf("ERROR: Point direction has changed at nc
         exit(-1);
        }
      }
```

Convex check. Type of function is void, no return value,

C v Python

- Try

Lec17_polyarea.c

- Modify to avoid error on concave/to compile from separate files.

```
TAHMac[1554] cc  Lec17_poly_area.c -o Lec17_poly_area
TAHMac[1555] Lec17_poly_area < poly_input.txt
=== Computing polygon area ==
Enter space-separated pairs of coordinates of the nodes, one pair per line.
Polygon has 5 nodes.
Area = 4.000000
```

```
TAHMac[1556] cat poly_input.txt
0 0
0 2
2 2
2 0
0 0
TAHMac[1557]
```

MIT OpenCourseWare

https://ocw.mit.edu

12.010 Computational Methods of Scientific Programming, Fall 2024