**Racing condition solution:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void \*functionC();

pthread\_mutex\_t mutex1 = PTHREAD\_MUTEX\_INITIALIZER;

int counter = 0;

main()

{

int rc1, rc2;

pthread\_t thread1, thread2;

/\* Create independent threads each of which will execute functionC \*/

if( (rc1=pthread\_create( &thread1, NULL, &functionC, NULL)) )

{

printf("Thread creation failed: %d\n", rc1);

}

if( (rc2=pthread\_create( &thread2, NULL, &functionC, NULL)) )

{

printf("Thread creation failed: %d\n", rc2);

}

/\* Wait till threads are complete before main continues. Unless we \*/

/\* wait we run the risk of executing an exit which will terminate \*/

/\* the process and all threads before the threads have completed. \*/

pthread\_join( thread1, NULL);

pthread\_join( thread2, NULL);

exit(0);

}

void \*functionC()

{

pthread\_mutex\_lock( &mutex1 );

counter++;

printf("Counter value: %d\n",counter);

pthread\_mutex\_unlock( &mutex1 );

}

**Memory allocation in C program,**

#include<stdio.h>

#include<stdlib.h>

//Function to compare unfrobnicated words

int frobcmp(char const \*a, char const \*b)

{

for(;; a++, b++) //Loop to look at entire words

{

if(\*a == ' ' && \*b == ' ') { return 0; }

else if (\*a == ' ' || ((\*a^42) < (\*b^42))) { return -1; }

else if (\*b == ' ' || ((\*a^42) > (\*b^42))) { return 1; }

}

}

//Function to use within qsort without having to cast

int cmp(const void\* in1, const void\* in2)

{

//We cast to pointers to pointers since thats what our

//words array holds

const char\* a = \*(const char\*\*)in1;

const char\* b = \*(const char\*\*)in2;

return frobcmp(a,b);

}

//Function to check for reading error

void readErr()

{

if(ferror(stdin))

{

fprintf(stderr, "Error while reading file!");

exit(1);

}

}

int main(void)

{

char\* word; //Holds one word at a time (delimited by spaces)

char\*\* words; //Array to hold pointers to words

word = (char\*)malloc(sizeof(char));

words = (char\*\*)malloc(sizeof(char\*));

//curr and next act as current and next iterators to use for noting

//EOF and auto adding spaces at the end of files

char curr = getchar();

readErr();

char next = getchar();

readErr();

int letterIterator = 0;

int wordsIterator = 0;

while(curr != EOF && !ferror(stdin)) //Read file until EOF

{

word[letterIterator] = curr; //Add letters to the word

//Constantly reallocate space for growing words

char\* temp = realloc(word, (letterIterator+2)\*sizeof(char));

if(temp != NULL)

{

//Make the word equal to the reallocated space

word = temp;

}

else //Allocation error, print error and exit

{

free(word);

fprintf(stderr, "Error Allocation Memory!");

exit(1);

}

if(curr == ' ') //Hit the end of the word

{

words[wordsIterator] = word; //Add word to words list

//Constantly reallocate space for growing wordslist

char\*\* anotherOne = realloc(words, (wordsIterator+2)\*sizeof(char\*));

if(anotherOne != NULL)

{

//Make words equal to reallocated space

words = anotherOne;

wordsIterator++;

//Set word back to empty by pointing it to other space

word = NULL;

word = (char\*)malloc(sizeof(char));

letterIterator = -1; //-1 to bring back to 0 when summed later

}

else //Allocation error, print error and exit

{

free(words);

fprintf(stderr, "Error Allocation Memory!");

exit(1);

}

}

if(next == EOF && curr == ' ')

{

break;

}

else if (curr == ' ' && next == ' ') //Ignore Extra Spaces

{

while(curr == ' ')

{

curr = getchar();

readErr();

}

next = getchar();

readErr();

letterIterator++;

continue;

}

else if(next == EOF) //Add a space at the end if there isn't already one

{

curr = ' ';

letterIterator++;

continue;

}

//increment our letter counter and get the next character

curr = next;

next = getchar();

readErr();

letterIterator++;

}

//Sort the frobnicated words from our words list

qsort(words, wordsIterator, sizeof(char\*), cmp);

//Output the words to STDOUT using putchar

for(size\_t i = 0; i < wordsIterator; i++)

{

for(size\_t j = 0; ;j++)

{

//EOF error checking

if(putchar(words[i][j]) == EOF)

{

fprintf(stderr, "Error while writing character!");

exit(1);

}

if(words[i][j] == ' ')

{

break;

}

}

}

//De-allocate all the space taken up for the words

for(size\_t i = 0; i < wordsIterator; i++)

{

free(words[i]);

}

free(words);

exit(0);

}

**phtread example**

#include <stdio.h>

#include <pthread.h>

#define NTHREADS 10

void \*thread\_function(void \*);

pthread\_mutex\_t mutex1 = PTHREAD\_MUTEX\_INITIALIZER;

int counter = 0;

main()

{

pthread\_t thread\_id[NTHREADS];

int i, j;

for(i=0; i < NTHREADS; i++)

{

pthread\_create( &thread\_id[i], NULL, thread\_function, NULL );

}

for(j=0; j < NTHREADS; j++)

{

pthread\_join( thread\_id[j], NULL);

}

printf("Final counter value: %d\n", counter);

}

void \*thread\_function(void\* dummyPtr)

{

counter++;

printf("Counter value for thread\_num: %ld is:%d\n", counter);

}

**Thread user, sys and real time comparison**

Performance Comparison

a. no — parallel option

$ time -p sort -g random.txt > /dev/null

real 40.64

user 216.58

sys 0.54

b. 1 thread

[classqin@lnxsrv07 ~/lab7]$ time -p sort -g --parallel=1 random.txt > /dev/null

real 183.46

user 183.11

sys 0.34

c. 2 threads

[classqin@lnxsrv07 ~/lab7]$ time -p sort -g --parallel=2 random.txt > /dev/null

real 98.58

user 188.57

sys 0.37

d. 4 threads

[classqin@lnxsrv07 ~/lab7]$ time -p sort -g --parallel=4 random.txt > /dev/null

real 57.67

user 196.92

sys 0.43

e. 8 threads

[classqin@lnxsrv07 ~/lab7]$ time -p sort -g --parallel=8 random.txt > /dev/null

real 36.47

user 199.40

sys 0.51

**Mutiphread in the project.**

#include "raymath.h"

#include "shaders.h"

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include <math.h>

#include <pthread.h>

static double dirs[6][3] =

{ {1,0,0}, {-1,0,0}, {0,1,0}, {0,-1,0}, {0,0,1}, {0,0,-1} };

static const int opposites[] = { 1, 0, 3, 2, 5, 4 };

static void

add\_sphereflake( scene\_t\* scene, int sphere\_id, int parent\_id, int dir,

double ratio, int recursion\_level )

{

sphere\_t\* parent = &scene->spheres[parent\_id];

sphere\_t\* child = &scene->spheres[sphere\_id];

/\* start at parents origin \*/

mul( child->org, dirs[dir], (1.+ratio)\*parent->rad );

add( child->org, child->org, parent->org );

child->rad = parent->rad \* ratio;

copy( child->color, parent->color );

child->shader = parent->shader;

scene->sphere\_count++;

}

static int

recursive\_add\_sphereflake( scene\_t\* scene, int parent\_id, int parent\_dir,

int sphere\_id, int dir,

int recursion\_level, int recursion\_limit )

{

const double ratio = 0.35;

add\_sphereflake( scene, sphere\_id, parent\_id, dir, ratio, recursion\_level );

if( recursion\_level > recursion\_limit )

{

return sphere\_id + 1;

}

/\* six children, one at each cardinal point \*/

parent\_id = sphere\_id;

sphere\_id = sphere\_id + 1;

for( int child\_dir=0; child\_dir<6; ++child\_dir )

{

/\* skip making spheres inside parent \*/

if( parent\_dir == opposites[child\_dir] ) continue;

sphere\_id = recursive\_add\_sphereflake( scene, parent\_id, parent\_dir,

sphere\_id, child\_dir,

recursion\_level + 1,

recursion\_limit );

}

return sphere\_id;

}

static scene\_t

create\_sphereflake\_scene( int recursion\_limit )

{

scene\_t scene;

Vec3 color;

sphere\_t\* sphere;

init\_scene( &scene );

// Pantone UC Gold 122

add\_light( &scene, 2, 5, 0, 0.996, 0.733, 0.212 );

// Pantone UCLA Blue (50,132,191)

add\_light( &scene, -5, 3, -5, 0.196, 0.517, 0.749 );

int max\_sphere\_count = 2 + powl( 6, recursion\_limit + 2 );

scene.spheres = realloc( scene.spheres,

max\_sphere\_count\*sizeof( sphere\_t ) );

if( !scene.spheres )

{

fprintf( stderr, "Failed to get memory for sphereflake. aborting.\n" );

exit( -1 );

}

// sphere = &(scene.spheres[0]);

// set( sphere->org, -0.5, -1.0, 0 );

// sphere->rad = 0.75;

// set( color, 0.85, 0.25, 0.25 );

// copy( sphere->color, color );

// sphere->shader = mirror\_shader;

/\* center sphere is special, child inherent shader and color \*/

sphere = &(scene.spheres[0]);

scene.sphere\_count++;

set( sphere->org, 0, -1, 0 );

sphere->rad = 0.75;

set( color, 0.75, 0.75, 0.75 );

copy( sphere->color, color );

sphere->shader = mirror\_shader;

recursive\_add\_sphereflake( &scene,

0, /\* parent is the first sphere \*/

-1, /\* -1 means no dir, make all children \*/

1, /\* next free sphere index \*/

2, /\* starting dir \*/

0, /\* starting recursion level \*/

recursion\_limit );

return scene;

}

static void

free\_scene( scene\_t\* arg )

{

free( arg->lights );

arg->light\_count = 0;

free( arg->spheres );

arg->sphere\_count = 0;

}

/\*\*\*\*\*\*

\* Constants that have a large effect on performance \*/

/\* how many levels to generate spheres \*/

enum { sphereflake\_recursion = 3 };

/\* output image size \*/

enum { height = 131 };

enum { width = 131 };

/\* antialiasing samples, more is higher quality, 0 for no AA \*/

enum { halfSamples = 4 };

/\*\*\*\*\*\*/

/\* color depth to output for ppm \*/

enum { max\_color = 255 };

/\* z value for ray \*/

enum { z = 1 };

/\*Define scene as the global variable\*/

scene\_t scene;

**/\*Define the nthreads\*/**

int nthreads;

**/\*Number of colours to be the depth of our 3D array\*/**

#define NUM\_COLORS 3

**/\*Array that hold the value that need to be printed\*/**

float printVals[width][height][NUM\_COLORS];

**/\*Function to be multithreaded\*/**

**void \* mult\_threads(void \* NUM\_thread)**

{

Vec3 camera\_pos;

set( camera\_pos, 0., 0., -4. );

Vec3 camera\_dir;

set( camera\_dir, 0., 0., 1. );

const double camera\_fov = 75.0 \* (PI/180.0);

Vec3 bg\_color;

set( bg\_color, 0.8, 0.8, 1 );

const double pixel\_dx = tan( 0.5\*camera\_fov ) / ((double)width\*0.5);

const double pixel\_dy = tan( 0.5\*camera\_fov ) / ((double)height\*0.5);

const double subsample\_dx

= halfSamples ? pixel\_dx / ((double)halfSamples\*2.0)

: pixel\_dx;

const double subsample\_dy

= halfSamples ? pixel\_dy / ((double)halfSamples\*2.0)

: pixel\_dy;

/\* for every pixel \*/

for( int px=0; px<width; ++px )

{

const double x = pixel\_dx \* ((double)( px-(width/2) ));

for( int py=0; py<height; ++py )

{

const double y = pixel\_dy \* ((double)( py-(height/2) ));

Vec3 pixel\_color;

set( pixel\_color, 0, 0, 0 );

for( int xs=-halfSamples; xs<=halfSamples; ++xs )

{

for( int ys=-halfSamples; ys<=halfSamples; ++ys )

{

double subx = x + ((double)xs)\*subsample\_dx;

double suby = y + ((double)ys)\*subsample\_dy;

/\* construct the ray coming out of the camera, through

\* the screen at (subx,suby)

\*/

ray\_t pixel\_ray;

copy( pixel\_ray.org, camera\_pos );

Vec3 pixel\_target;

set( pixel\_target, subx, suby, z );

sub( pixel\_ray.dir, pixel\_target, camera\_pos );

norm( pixel\_ray.dir, pixel\_ray.dir );

Vec3 sample\_color;

copy( sample\_color, bg\_color );

/\* trace the ray from the camera that

\* passes through this pixel \*/

trace( &scene, sample\_color, &pixel\_ray, 0 );

/\* sum color for subpixel AA \*/

add( pixel\_color, pixel\_color, sample\_color );

}

}

/\* at this point, have accumulated (2\*halfSamples)^2 samples,

\* so need to average out the final pixel color

\*/

if( halfSamples )

{

mul( pixel\_color, pixel\_color,

(1.0/( 4.0 \* halfSamples \* halfSamples ) ) );

}

/\* done, final floating point color values are in pixel\_color \*/

float scaled\_color[3];

scaled\_color[0] = gamma( pixel\_color[0] ) \* max\_color;

scaled\_color[1] = gamma( pixel\_color[1] ) \* max\_color;

scaled\_color[2] = gamma( pixel\_color[2] ) \* max\_color;

/\* enforce caps, replace with real gamma \*/

for( int i=0; i<3; i++)

scaled\_color[i] = max( min(scaled\_color[i], 255), 0);

/\* write this pixel out to disk. ppm is forgiving about whitespace,

\* but has a maximum of 70 chars/line, so use one line per pixel

\*/

**printVals[px][py][0] = scaled\_color[0];**

**printVals[px][py][1] = scaled\_color[1];**

**printVals[px][py][2] = scaled\_color[2];**

**}**

}

return NULL;

}

int

main ( int argc, char \*\*argv)

{

**nthreads = argc = 2 ? atoi ( argv[1]) : 0 ;**

**if(nthreads < 1)**

{

fprintf( stderr, "%s: usage: %s NTHREADS\n", argv[0], argv[0] );

return 1;

}

scene = create\_sphereflake\_scene ( sphereflake\_recursion );

/\* Write the image format header \*/

/\* P3 is an ASCII-formatted, color, PPM file \*/

printf( "P3\n%d %d\n%d\n", width, height, max\_color );

printf( "# Rendering scene with %d spheres and %d lights\n",

scene.sphere\_count,

scene.light\_count );

**/\*Create threads\*/**

pthread\_t threads[nthreads];

/\*Create thread ID array\*/

int thread\_id[nthreads];

for(int t = 0; t < nthreads; t++)

{

thread\_id[t] = t;

int ret = pthread\_create(&threads[t], NULL, mult\_threads,&thread\_id[t]);

if(ret)

{

fprintf(stderr, "Error creating threads. Error code %d\n",ret);

exit(-1);

}

}

/**\*Free the threds\*/**

for(int t = 0; t < nthreads; t++)

{

int ret = pthread\_join(threads[t], NULL);

if(ret)

{

fprintf(stderr, "Error freeing threads. Error code %d\n", ret);

exit(-1);

}

}

/**\*print the stuff\*/**

for(int i = 0; i < width; i++)

{

for(int j = 0; j < height; j++)

{

printf("%.0f %.0f %.0f\n",

printVals[i][j][0], printVals[i][j][1], printVals[i][j][2]);

}

printf("\n");

}

free\_scene( &scene );

if( ferror( stdout ) || fclose( stdout ) != 0 )

{

fprintf( stderr, "Output error\n" );

return 1;

}

return 0;

}

**system call**

buffered I/O

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main (int argc, char \*argv[])

{

//check number of arugments

if(argc != 3)

{

fprintf(stderr, "Please input the correct arguments.\n");

exit(1);

}

//check the length of arugement

char\* from= argv[1];

char\* to= argv[2];

if(strlen(from)!=strlen(to))

{

fprintf(stderr, "Two operands are not the same lenght.\n");

exit(1);

}

//check dupulication

for(size\_t i = 0; i < strlen(from); ++i)

{

for( size\_t j = i+1; j < strlen(from); ++j)

{

if(from[j]==from[i])

{

fprintf(stderr, "this operand has duplicated characters");

exit(1);

}

}

}

//replace the first operand for second operand

char current= getchar();

int match = 0;

while(current != EOF)

{

for(size\_t i =0; i<strlen(from); ++i)

{

if(current= from[i])

{

putchar(to[i]);

match=1;

break;

}

else

{

match=0;

}

}

if(!match)

{

putchar(current);

}

current = getchar();

}

return 0;

}

**Unbuffered Version**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

int main (int argc, char \*argv[])

{

//check number of arugments

if(argc != 3)

{

fprintf(stderr, "Please input the correct arguments.\n");

exit(1);

}

//check the length of arugement

char\* from= argv[1];

char\* to= argv[2];

if(strlen(from)!=strlen(to))

{

fprintf(stderr, "Two operands are not the same lenght.\n");

exit(1);

}

//check dupulication

for(size\_t i = 0; i < strlen(from); ++i)

{

for( size\_t j = i+1; j < strlen(from); ++j)

{

if(from[j]==from[i])

{

fprintf(stderr, "this operand has duplicated characters");

exit(1);

}

}

}

//replace the first operand for second operand

char current [1];

ssize\_t ret = read(0,current,1);

int match =0 ;

while (ret > 0)

{

for ( size\_t i =0; i< strlen(from);++i)

{

if(current[0]== from[i])

{

current[0]=to[i];

write(1 ,current,1);

match =1;

break;

}

else

{

match=0;

}

}

if (!match)

{

write (1 ,current ,1);

}

ret = read (0, current, 1);

}

return 0;

}

Test time:

tr2b(buffered version):

[classqin@lnxsrv07 ~/lab6]$ time ./tr2b 'abc' 'def' < input\_c.txt > output\_tr2b

real 0m0.186s

user 0m0.124s

sys 0m0.005s

tr2u(unbuffered version):

[classqin@lnxsrv07 ~/lab6]$ time ./tr2u 'abc' 'def' < input\_c.txt > output\_tr2u

real 0m6.526s

user 0m0.282s

sys 0m6.201s

**lab 6**

**Sys Call**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/stat.h>

int comparison = 0;

//function to compare unfrobnicared words

int frobcmp (char const \*a, char const \*b)

{

for(;; a++, b++) // go to the entire words

{

if(\*a == ' ' && \*b == ' ' ) {return 0;}

else if(\*a == ' ' || ((\*a^42) < (\*b^42)) ) {return -1;}

else if(\*b == ' '|| ((\*a^42) > (\*b^42)) ) {return 1;}

}

}

//function to use within qsort without having to cast

int cmp (const void \* n1, const void \* n2)

{

//casting pointers to pointers

const void \* a = \*(char const\*\*)n1;

const void \* b = \*(char const\*\*)n2;

comparison++;

return frobcmp(a,b);

}

//function to check for reading error

void checkerr(ssize\_t s)

{

if(s < 0)

{

fprintf(stderr,"reading error!");

exit(1);

}

}

int main (void)

{

struct stat start;

if(fstat(0, &start)<0)

{

fprintf(stderr, "Error!");

}

char\* word;

char\*\* words;

size\_t wordsiterator=0;

if(S\_ISREG(start.st\_mode))

{

word=(char\*)malloc(start.st\_size\*sizeof(char));// allocated enough memeoy

ssize\_t s = read(0, word,start.st\_size);

int ptrcount=0;

int count =1;

if(s>0)//check error

{

for(size\_t i=0; i<start.st\_size;i+=count)

{

count = 1;

if(i == start.st\_size -1)

{

word[i] = ' ';

}

if(word[i] == ' ')

{

for(size\_t k= i;word[k]!= ' ';k++)

{

count++;

}

ptrcount++;

}

}

}

words=(char\*\*)malloc(ptrcount\*sizeof(char\*));

int pin =0;

for(size\_t i=0; i<start.st\_size; i++)

{

if(pin==0 && word[i]!= ' ')

{

words[wordsiterator] = &word[i];

wordsiterator++;

pin=1;

}

else if (pin ==1 && word [i] == ' ')

{

pin = 0;

}

}

}

else

{

words = (char\*\*)malloc(sizeof(char\*));

}

char\* word\_h;// hold one word at a time

word\_h = (char\*)malloc(sizeof(char));

//current and next are iterators to use for indicating

//EOF and auto adding spaces the end of file

char current[1];

ssize\_t curpos = read(0, current, 1);

checkerr(curpos);

char next[1];

ssize\_t nextpos = read(0, next, 1);

checkerr(nextpos);

int letteriterator=0;

while ( curpos >0)

{

word\_h[letteriterator] = current[0]; // add letters to the words

char\* temp\_memo = realloc(word\_h, (letteriterator+2)\*sizeof(char));

if ( temp\_memo != NULL)

{

word\_h= temp\_memo;

}

else //allocating error print error and then exit

{

free(word\_h);

fprintf(stderr, "allocating error!");

exit(1);

}

if (current[0] == ' ')// at the end of the word

{

words[wordsiterator] = word\_h; // add word to the words array

char\*\* temp1 = realloc(words, (wordsiterator+2)\*sizeof(char\*));

if(temp1 != NULL)

{

words=temp1;//make sures words equal to reallocated spaces

wordsiterator++;

//let the word points to other space

word\_h=NULL;

word\_h =(char\*)malloc(sizeof(char));

letteriterator = -1; //-1 back to 0

}

else //allocating error print error and then exit

{

free(words);

fprintf(stderr, "allocating error!");

exit(1);

}

}

if(nextpos == 0 && current[0] == ' ')

{

break;

}

else if ( current[0] == ' ' && next[0] == ' ' ) //skip extra spaces

{

while(current[0] == ' ')

{

curpos = read(0,current, 1);

checkerr(curpos);

}

nextpos = read(0,next,1);

checkerr(nextpos);

letteriterator++;

continue;

}

else if ( nextpos == 0 ) // if next is at the end of the file

{

current[0] = ' ';

letteriterator++;

continue;

}

current[0] = next[0];

nextpos = read (0, next, 1);

checkerr(nextpos);

letteriterator++;

}

// sort the frobnicated word from the words array

qsort(words, wordsiterator, sizeof(char\*), cmp);

// output the words to stdoutt using puchar

for(size\_t i = 0; i < wordsiterator; i++)

{

long word\_size= 0;

for(size\_t j = 0;; j++)

{

word\_size++;

if (words[i][j]== ' ')

{

break;

}

}

if (write(1, words[i],word\_size) == 0)

{

fprintf(stderr, "writing error!");

exit(1);

}

}

fprintf(stderr, "Comparison: %i\n", comparison);

free(words);

exit(0);

}

**Dynamic link in the make file**

OPTIMIZE = -O2

CC = gcc

CFLAGS = $(OPTIMIZE) -g3 -Wall -Wextra -march=native -mtune=native -mrdrnd

randlibsw.so:

$(CC) $(CFLAGS) -fPIC -c randlibsw.c -o randlibsw.o

$(CC) $(CFLAGS) -shared -o randlibsw.so randlibsw.o

randlibhw.so:

$(CC) $(CFLAGS) -fPIC -c randlibhw.c -o randlibhw.o

$(CC) $(CFLAGS) -shared -o randlibhw.so randlibhw.o

randmain:

$(CC) $(CFLAGS) -c randcpuid.c -o randcpuid.o

$(CC) $(CFLAGS) -c randmain.c -o randmain.o

$(CC) $(CFLAGS) -ldl -Wl,-rpath=$(PWD) randmain.o randcpuid.o -o randmain

Simple Python program .

#!/usr/bin/python

import sys

def DataTypes(dummy):

#Integers & Floats

tempInt = 5

tempFloat = 4.5

#Strings

sampleSTR = "My first python script!"

print sampleSTR

sampleSTR = sampleSTR + 'A' + sampleSTR \* 2

# + for joining strings and \* repeating

sampleSTR[0]

sampleSTR[5]

len(sampleSTR)

sampleSTR[3:7]

sampleSTR[3:len(sampleSTR)]

sampleSTR[:5]

sampleSTR[5:]

sampleSTR[-1]

#Immutable!Cannot change its members

#sample[5] = 'd'

#Lists are mutable

lst = [10, 20, 'hello', 'world']

lst[0]

lst[3]

lst[:]

lst[2] = 30

lst.append('hello')

lst[:] = []

'''

MultiLine Comment

Tuples are immutable

cannot assign values like tup[0]=6

'''

tup = (1,2,3,4)

#Sets - No duplicate elements. Good for membership testing and finding duplicates

s1 = set('abracadabra')

s2 = set('alacazam')

print 'The sets contain - ', s1, s2

#Supports set operations

s1 - s2 #elements in s1 but not in s2

s1 | s2 # OR operation

s1 & s2 # AND operation

s1 ^ s2 # XOR operation

#Converting a list to a set with unique elements, and back to a list again

basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

uniquebasket = list(set(basket))

#Dictionary: Associative arrays. for having (key,value) pairs. Keys are unique.

d = {}

d[12345] = "Tom"

d[67899] = "SeasHelp"

#Unique keys

d[911] = "Emergency"

d.values()

d.keys()

d.items()

print d

if 'Tom' in d.values():

print 'Tom Present!'

elif 'SeasHelp' in d.values():

print 'Seas present!'

else:

print 'None'

temp = 5

if temp == 5:

print temp

def Looping(forloops, whileloops):

a = 0

while a < 10:

print 'The current value of a is', a

print 'The current value of a is {0} and a+1 is {1}'.format(a, a+1)

a = a + 1

for i in range(0, 10):

print 'The current value of i is', i

lst = ['a', 'b', 'c', 'd']

for i in range(len(lst)):

print i, lst[i]

range(3, 10)

#Generates a list [3, 4, 5, 6, 7, 8, 9]

#Increments of 2

range(0, 10, 2)

#Generates list [0, 2, 4, 6, 8]

return lst

class Engine:

modelID=0

#Constructor with a reference to self - Like the "this" pointer in c++

def \_\_init\_\_(self, ID):

self.modelID = ID

def printID(self):

print 'Engine model ID is ', self.modelID

def foo(self, i):

print i

def main():

#sys.argv for accessing command line arguments

print sys.argv

Looping(5, 5)

#Named arguments

Looping(whileloops=6, forloops=10)

DataTypes([5, 4, 5, 'arguments'])

myEngine = Engine(20)

myEngine.printID()

print 'You can access the internal members directly. modelID is ', myEngine.modelID

#Defining an entry point for the script

if \_\_name\_\_=="\_\_main\_\_":

main()

**lab4 python**

#!/usr/bin/python

""""

This file was created by Jahan Kuruvilla Cherian on 01/21/16.

This file runs the POSIX command comm to compare two files and output

the differences and similarities in the files. This modified version also

supports unsorted comparisons.

"""

import sys, string, locale

from optparse import OptionParser

class compare:

def \_\_init\_\_(self,filename1, filename2):

try:

if filename1 == "-":

f1 = sys.stdin

f2 = open(filename2, 'r')

elif filename2 == "-":

f2 = sys.stdin

f1 = open(filename1, 'r')

elif filename1 == "-" and filename2 == "-":

print("Error! Can't read both files from STDIN")

exit()

else:

f1 = open(filename1, 'r')

f2 = open(filename2, 'r')

self.lines1 = f1.read().split('\n')

self.lines2 = f2.read().split('\n')

del self.lines1[len(self.lines1)-1]

del self.lines2[len(self.lines2)-1]

self.list1 = []

self.list2 = []

self.list3 = []

f1.close()

f2.close()

except IOError as e:

errno = e.errno

strerror = e.strerror

parser.error("I/O error({0}): {1}".

format(errno, strerror))

def add\_Newline(self, file):

for i in range(len(file)):

if file[i] == '':

file[i] = '\n'

if file[i].count(' ') >= 1 and file[i].isspace():

file[i] = '\n' \* file[i].count(' ')

def modify(self):

self.add\_Newline(self.lines1)

self.add\_Newline(self.lines2)

def u\_comparison(self):

self.modify()

for i in range(len(self.lines1)):

for j in range(len(self.lines2)):

if self.lines1[i] == self.lines2[j]:

self.list3.append(self.lines1[i])

self.list1.append(" ")

self.list2.append(" ")

del self.lines2[j]

similar = True

break

else:

similar = False

if similar == False:

self.list1.append(self.lines1[i])

self.list2.append("")

self.list3.append("")

extra\_spaces = [" "] \* len(self.lines2)

extra\_null = [""] \* len(self.lines2)

self.list1 += extra\_spaces

self.list3 += extra\_null

self.list2 += self.lines2

def is\_Sorted(self, file, file\_num):

for i in range(len(file) - 1):

if locale.strcoll(file[i],file[i+1]) > 0:

print ("File %s is not sorted") % file

exit()

return True

def s\_comparison(self):

if self.is\_Sorted(self.lines1, 1) and self.is\_Sorted(self.lines2, 2):

self.modify()

i = j = 0

while i < len(self.lines1) and j < len(self.lines2):

if self.lines1[i] == self.lines2[j]:

self.list3.append(self.lines1[i])

self.list1.append(" ")

self.list2.append(" ")

self.lines1[i] = ''

self.lines2[j] = ''

i += 1

j += 1

elif self.lines1[i] > self.lines2[j]:

self.list2.append(self.lines2[j])

self.list1.append(" ")

self.list3.append("")

self.lines2[j] = ''

j += 1

elif self.lines1[i] < self.lines2[j]:

self.list1.append(self.lines1[i])

self.list2.append("")

self.list3.append("")

self.lines1[i] = ''

i +=1

if i > j:

extra\_spaces = [" "] \* len(self.lines2)

extra\_null = [""] \* len(self.lines2)

self.list2 += self.lines2

self.list1 += extra\_spaces

self.list3 += extra\_null

elif i < j:

extra\_spaces = [""] \* len(self.lines1)

self.list1 += self.lines1

self.list2 += extra\_spaces

self.list3 += extra\_spaces

def printer(self, option1, option2, option3 ):

if option1 == True:

self.list1 = [''] \* len(self.list1)

if option2 == True:

self.list2 = [''] \* len(self.list2)

if option3 == True:

self.list3 = [''] \* len(self.list3)

final\_list = []

for i in range(len(self.list1)):

final\_list.append(self.list1[i] + self.list2[i] + self.list3[i])

for j in range(len(final\_list)):

for k in range(len(final\_list[j])):

if final\_list[j][k] != " ":

if final\_list[j][k] == "\n":

str = final\_list[j]

for i in range(len(str)):

if str[i] == '\n':

sys.stdout.write(" ")

else:

sys.stdout.write(str[i])

print('')

break

else:

print(final\_list[j])

break

def main():

locale.setlocale(locale.LC\_ALL, 'C')

version\_msg = "%prog 1.0"

usage\_msg = """%prog [OPTION]... FILE1 FILE2

Output the comparison results between FILE1 and FILE2 in three columns."""

parser = OptionParser(version=version\_msg, usage=usage\_msg)

parser.add\_option("-1", action="store\_true", dest="sup1",

default = False,

help="suppress column 1 (lines unique to FILE1)")

parser.add\_option("-2", action="store\_true", dest="sup2",

default = False,

help="suppress column 2 (lines unique to FILE2)")

parser.add\_option("-3", action="store\_true", dest="sup3",

default = False,

help="suppress column 3 (lines that appear in both files)")

parser.add\_option("-u", action="store\_true", dest="unsort",

default = False,

help="run comparison on unsorted files line by line." )

options, args = parser.parse\_args(sys.argv[1:])

try:

sup1 = bool(options.sup1)

sup2 = bool(options.sup2)

sup3 = bool(options.sup3)

unsort = bool(options.unsort)

except:

parser.error("invalid option type: {0}".

format(options.sup1))

if len(args) != 2:

parser.error("missing option arguments")

input\_file1 = args[0]

input\_file2 = args[1]

try:

comparator = compare(input\_file1, input\_file2)

if unsort:

comparator.u\_comparison()

else:

comparator.s\_comparison()

comparator.printer(sup1, sup2, sup3)

except IOError as e:

errno = e.errno

strerror = e.strerror

parser.error("I/O error({0}): {1}".

format(errno, strerror))

if \_\_name\_\_ == "\_\_main\_\_":

main()